



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT – I-Rich Internet Applications – SCS1401**

## **I. Rich Internet Application Overview**

Introduction to Web2.0 - Key characteristics of Rich Internet Application - Current Rich Internet Application platforms - Rich Internet Application benefits - Rich Internet Application patterns and best practices - Rich Internet Application architecture - Restful Web Services with Nodes.

### **Introduction to Web 2.0**

#### **Web 1.0**

- Web 1.0 refers to the first stage in the World Wide Web
- Entirely made up of web pages connected by hyperlinks.
- A set of static websites that were not yet providing interactive content.
- Used as “Information portal”.

**Examples:** Amazon, Yahoo, Personal web pages

#### **Web 2.0**

- Web 2.0 is the term used to describe a variety of web sites and applications that allow anyone to create and share online information or material they have created.
- It allows people to create, share, collaborate & communicate.
- Allows everyone to produce their content.
- Gives the users the possibility to control their data.
- **Web 2.0** allows groups of people to work on a document or spreadsheet simultaneously.
- In the background a computer keeps track of who made what changes where and when.
- **Web-based applications** can be accessed from anywhere.
- **Web 2.0 Examples**
  - Web applications ( Google Docs, Flickr)
  - Video sharing sites (YouTube)
  - Wikis (Media Wiki)
  - Blogs (WordPress)
  - Social networking (Facebook)
  - Microblogging (Twitter)
  - Hosted services (Google Maps)

## Web 3.0

- It refers to the evolution of web utilization and interaction which includes altering the Web into a database.
- It enables the upgradation of back-end of the web, after a long time of focus on the front-end
- Data isn't owned but instead shared, where services show different views for the same web / the same data.
- The Semantic Web (3.0) promises to establish "the world's information" in more reasonable way than Google can ever attain with their existing engine schema.
- The Semantic Web necessitates the use of a declarative ontological language like OWL to produce domain-specific ontologies that machines can use to reason about information and make new conclusions.
- **Examples:** Online Coupons, Voice Search, FB app

## Difference between Web 1.0, Web 2.0 and Web 3.0

The difference between web 1.0,2.0,3.0 are discussed in the figure 1.1.

<b>Different between WEB 3.0 with WEB 2.0 and WEB 1.0</b>		
<b>WEB 1.0</b>	<b>WEB 2.0</b>	<b>WEB 3.0</b>
The web	The social web	The semantic web
Read only web	Read and write web	Read, write and execute web
Information sharing	Interaction	Immersion
Connect information	Connect people	Connect knowledge
All about static content, one way publishing (one way communication)	More about two way communication through social networking, blogging, tagging and wikis.	Curiously undefined.
Example : Personal web sites	Example : Blogs, Facebook	Example : Semantic blog (semiblog, haystack)

**Figure 1.1 Difference between Web 1.0, Web 2.0, Web 3.0**

## What is RIA?

- Rich Internet Applications (RIAs) are web applications that have the features and functionality of traditional desktop applications.

- RIAs typically provide a “no-refresh” look to the user interface.
- RIA provides HDuX – High Definition User eXperience.

### **Limitations of HTML based Web Applications**

- **Process Complexity** – Multi step tasks are time consuming and frustrating.
- **Data Complexity** – HTML based web applications don't facilitate the manipulation and visualization of complex data.
- **Feedback Complexity** – Server based processing limits the scope of an interactive user experience.

### **What is special in RIA?**

- RIA works in Web.
- RIA appears – never refreshes.
- RIA reduces network traffic.
- RIA is rich.
- RIA makes it easy.

### **History of RIA**

- The term "rich Internet application" was introduced in a white paper in March 2002 by Macromedia (now merged into Adobe).
- The concept had existed earlier under names such as:
  - Remote Scripting, by Microsoft, circa 1999
  - X Internet, by Forrester Research in October 2000
  - Rich (Web) clients
  - Rich Web application

## Differences between a Desktop, Traditional Web Application and RIA

The below figure 1.2 describes the difference between Desktop, Traditional Web Applications and Rich Internet Applications.

Features	Desktop	Web	RIA
Rich User Experience	Yes	No	Yes
Interactive, Responsive	Yes	No	Yes
Low Maintenance	No	Yes	Yes

**Figure 1.2 Difference between Desktop, Traditional Web, RIA**

### RIA Tools

- **Adobe Flex** → a highly productive, open source application framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops and devices.
- **OpenLaszlo** → a open source platform for the development and delivery of Rich Internet Applications, consists of the LZX programming language and the OpenLaszlo Server.
- **Microsoft Silverlight** → a powerful development tool for creating engaging, interactive user experiences for Web and mobile applications.
- **JavaFX** → a software platform for creating and delivering desktop applications, as well as Rich Internet Applications (RIAs) that can run across a wide variety of devices.
- The logos of RIA tools are shown in figure 1.3.



**Figure 1.3 RIA Tools**

### **Platforms of RIA**

- Adobe Flash
- Java
- Microsoft Silverlight
- AJAX

### **Adobe Flash**

- Manipulates vector and raster graphics to provide animation of text, drawings, and still images.
- Supports bidirectional streaming of audio and video.
- It can capture user input via mouse, keyboard, microphone, and camera.
- Flash contains an object-oriented language called ActionScript and supports automation via the JavaScript Flash language (JSFL).
- Flash content may be displayed on various computer systems and devices, using Adobe Flash Player, some mobile phones and a few other electronic devices (using Flash Lite).

### **Java**

- Java applets are used to create interactive visualization.
- To present video, three dimensional objects and other media.

- Java applets are more appropriate for complex visualizations that require significant programming effort in high level language or communications between applet and originating server.
- JavaFX is considered as another competitor for Rich Internet Applications.

### Microsoft Silverlight

- It has emerged as a potential competitor to Flash.
- To provide video streaming for many high profile events, including the 2008 Summer Olympics in Beijing, the 2010 Winter Olympics in Vancouver, and the 2008 conventions for both major political parties in the United States.
- Silverlight is also used by Netflix for its instant video streaming service.

### Characteristics of RIA

- Rich user experience
- Interactive- An RIA can use a wider range of controls that allow greater efficiency and enhance the user experience.
  - Users can interact directly with page elements through editing or drag-and-drop tools.
  - They can also do things like pan across a map or other image.
- Responsive
- Low maintenance
- Is plastic ( changing, transforming)
- Breaks walled gardens
- **Partial-page updating:** RIAs incorporate additional technologies, such as real-time streaming, high-performance client-side virtual machines, and local caching mechanisms that reduce latency (wait times) and increase responsiveness.
- **Better feedback:** Because of their ability to change parts of pages without reloading, RIAs can provide the user with fast and accurate feedback, real-time confirmation of actions and choices, and informative and detailed error messages.
- **Consistency of look and feel:** With RIA tools, the user interface and experience with different browsers and operating systems can be more carefully controlled and made consistent.
- **Offline use:** When connectivity is unavailable, it might still be possible to use an RIA if the app is designed to retain its state locally on the client machine.
- **Caching**
- RIA client has the ability of keeping the server information during a period of time improving application performance and UI responsiveness.

- **Security**
- RIAs should be as secure as any other web application, and the framework should be well equipped to enforce limitations appropriately when the user lacks the required privileges, especially when running within a constrained environment such as a sandbox.
- **Advanced Communication**
- Sophisticated communications with supporting servers through optimized network protocols can considerably enhance the user experience.
- **Rapid Development**

An RIA Framework should facilitate rapid development of a rich user experience through its easy-to-use interfaces in ways that help developers.

- **Improved Features**

RIA allow programmers to embed various functionalities in graphics-based web pages that look fascinating and engaging like desktop applications.

RIA provide complex application screens on which various mixed media, including different fonts, vector graphic and bitmap files online conferencing etc. are paused by using different modern development tools.

## **Advantages of RIA**

- **Remotely accessed application** – The basic principle of RIA is the ability to have any new user connect and run the application from any location as long as they are connected to the network.
- **Full interactive experience** – Unlike Web applications that provide page-by-page interaction and feedback, RIA provides a full interactive end-user experience.
- **Integrated Form Editor**
- **A Comprehensive Solution** – A comprehensive end-to-end solution facilitating full interactive distributed clients and a centralized managed server.
- **A Single Unified IDE & Paradigm** – The RIA is supported by a single and unified IDE and development paradigm for defining server-side and client-side logic along with the user interface design.
- **Automatic Logic Partitioning** – The RIA deployment modules facilitate optimized automatic logic partitioning between client and server.
- **Performance-Aware Development** – The Rich Client Studio is a performance-aware platform for developing response-optimized applications.
- **Native Look & Feel** – It provides an automatic reflection of the native look and feel of the selected client machine.



- **Browser free solution** – RIA is independent of browser vendors and browser versions.
- **Local Resources Activation** – RIA supports various activities that can be executed on the client side. For example:
  - OS command
  - File manipulation
  - OS environment manipulation
- **More Responsive:** The agile response from applications keeps the user engaged while improving user productivity.
- **Interactive User Interface:**
  - RIAs have more interactive UI as they can be used to provide information in more appealing way in lesser time as compared to conventional internet applications.
  - This fast interactivity with the application improves user satisfaction quite significantly.
  - **Less Internet Traffic and Faster Processing:**
    - Rich internet application does not refresh entire page and that leads to less traffic and faster processing.
- **Simplifying Online Transactions:**
  - RIAs eliminate the multi-page for multi-step transactions by presenting all pertinent information to users without leaving the initial environment.
  - This improves customer satisfaction and customer loyalty as they view the service provider as someone who understands their needs.
- **Easier Mobile Access to Information**

A number of manufacturing systems can potentially benefit from mobile access.

By using RIAs, plant managers and supervisors can access mission-critical information without being anchored to a static workstation.

Mobile devices can help management more quickly identify and resolve problems on the shop floor, locate inventory and update information, in real-time and on-location.
- **Better Data Visualization**

Manufacturers who have implemented RIAs find users suggesting new and useful ways of visualizing data all the time.

➤ Data visualization improves decision making and can reduce product costs and direct material spent by identifying bottlenecks, anticipating shortages and improving lean manufacturing procedures.

- **Batch and Item Level Tracking**
  - Tracking of inventory, supplies, spare parts, and even labor resources using RFID enhances business efficiency.
  - Use of RFID and custom applications to support batch and item level tracking in manufacturing processes can improve productivity for activities.
- **Integration Across Multiple Systems**
  - RIAs can pull data from multiple data warehouses and systems, and present information on a single screen or reduced number of screens with visualization capabilities, as well as to drill-down to necessary detail.

### **Business Benefits of RIA**

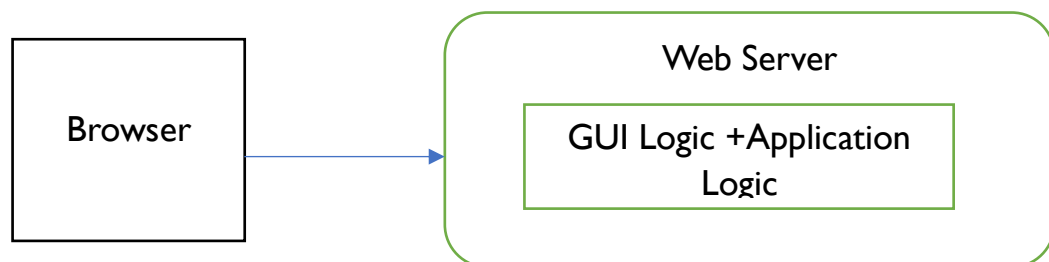
- Enables more transactions
- Retains customers
- Low operational cost
- Improves performance

### **Application Benefits:**

- Add value to your applications
- Meet your customer expectations
- Simplify & speed up processes
- Get regular and dedicated visitors
- Save costs on Bandwidth
- Increase your productivity

### **First Generation Web Applications**

- First generation web applications were page oriented.
- All GUI logic and application logic inside the same web page shown in Figure 1.4.

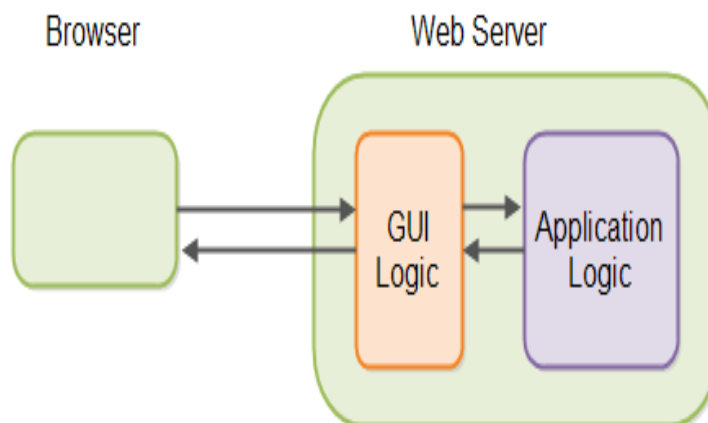


**Figure 1.4 First Generation of Web Applications**

- Every action the application allowed was typically embedded in its own web page script.
- Each script was like a separate transaction which executed the application logic, and generated the GUI to be sent back to the browser after the application logic was executed.
- First generation web page technologies include Servlets (Java), JSP (JavaServer Pages), ASP, PHP and CGI scripts in Perl etc.

## Second Generation Web Applications

- In second generation web applications developers found ways to separate the GUI logic from the application logic on the server shown in Figure 1.5.
- Web page scripts were used for the GUI logic
- Real classes and objects were used for the application logic
- Frameworks were developed to help make second generation web applications easier to develop.
- Examples of such frameworks are ASP.NET (.NET), Struts + Struts 2 (Java), Spring MVC (Java), JSF (JavaServer Faces), Wicket (Java) Tapestry (Java) and many others.
- GUI logic was written in the same language as the application logic, changing the programming language meant rewriting the whole application again.

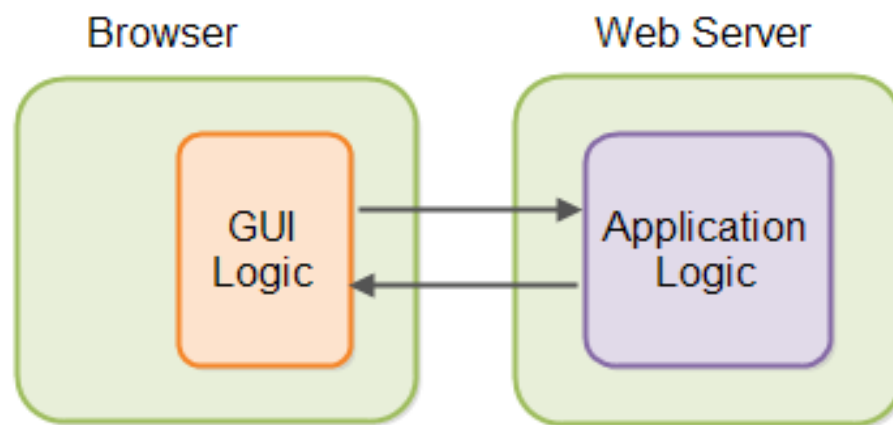


**Figure 1.5 Second Generation Web Applications**

## RIA Web Applications

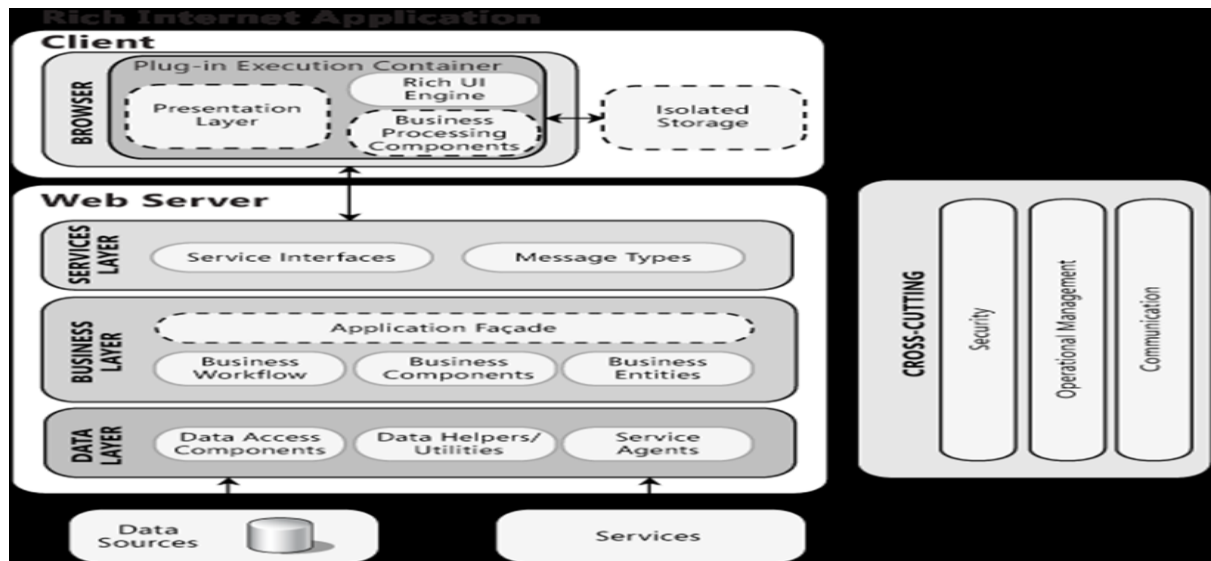
- RIA (Rich Internet Applications) web applications are the third generation of web applications.

- RIA technologies are typically executed in the browser using either JavaScript, Flash, JavaFX or Silverlight
- The GUI logic is now moved from the web server to the browser.
- GUI logic is executed in the browser, the CPU time needed to generate the GUI is lifted off the server, freeing up more CPU cycles for executing application logic.
- GUI state can be kept in the browser, thus further cleaning up the server side of RIA web applications.
- GUI logic is completely separated from the application logic is shown in Figure 1.6, it becomes easier to develop reusable GUI components
- No matter what server-side programming language is used for the application logic.



**Figure 1.6 RIA Web Applications**

## Architecture of RIA



**Figure 1.7 Architecture of RIA**

A typical Rich Internet Application is decomposed into three layers is shown in Figure 1.7.

- **Presentation layer** - contains UI and presentation logic components
- **Business layer** - contains business logic, business workflow and business entities components
- **Data layer** – Contains data access and service agent components.
- It is common in RIAs to move some of business processing and even the data access code to the client.
- The client may in fact contain some or all of the functionality of the business and data layers, depending on the application scenario.

### UI components

- Users can interact with the application
- Format data and render data to users
- Acquire and validate data

### Application facade

- To combine multiple business operations into single message-based operation
- The facade can be accessed from the presentation layer using a range of communication technologies.

### Data access logic components

- Abstract the logic needed to access the underlying data stores.
- This centralizes data access functionality, makes it easier to configure and maintain.

### **Data helpers/utilities**

- For centralizing generic data access functionality (managing db connection and caching data).
- Data source specific helper components can be designed to abstract the complexity of accessing the db.
- Service Agents – Basic mapping between the format of the data exposed by the service and the format your application needs.
- Business components – implement the business logic of the application. Implement business rules and perform business tasks.
- Business workflows – Define and coordinate long running, multistep business processes. They can be implemented using business process management tools.
- Design considerations
  - Choose an appropriate technology based on application requirements.( eg . Windows Forms, OBA, WPF)
  - Separate presentation logic from interface implementation
    - Eases maintenance
    - Promotes reusability
    - Improves testability
- Identify the presentation tasks and presentation flows
  - Helps to design each screen and each step-in multi-screen or wizard Processes.
- Design to provide a suitable and usable interface
  - Features like layout, navigation, choice of controls to maximize accessibility and usability.
- Extract business rules and other tasks not related to the interface.
- Reuse common presentation logic – (Libraries that contain templates, generalized client-side validation functions and helper classes)
- Loose couple your client from any remote services it uses.
  - Use a message-based interface to communicate with services located on separate physical tiers.

- Avoid tight coupling to objects in other layers – Use the abstract base classes or messaging when communicating with other layers of the application.
- Reduce round trips when accessing remote layers – Use coarse grained methods and execute them asynchronously to avoid blocking or freezing the UI.

## **General design considerations**

### **1.Choose a RIA based on audience, rich interface, and ease of deployment**

- Consider designing a RIA when your vital audience is using a browser that supports RIAs.
- If part of your vital audience is on a non-RIA supported browser, consider whether limiting browser choice to a supported version is a possibility.
- If you cannot influence the browser choice, consider if the loss of audience is significant enough to require choice of an alternative type of application, such as a Web application using AJAX.
- With a RIA, the ease of deployment and maintenance is similar to that of a Web application, assuming that your clients have a reliable network connection.
- RIA implementations are well suited to Web-based scenarios where you need visualization beyond that provided by basic HTML.
- They are likely to have more consistent behavior and require less testing across the range of supported browsers when compared to Web applications that utilize advanced functions and code customizations.
- RIA implementations are also perfect for streaming-media applications.
- They are less suited to extremely complex multipage UIs.

### **2.Design to use a Web infrastructure utilizing services.**

- RIA implementations require an infrastructure similar to Web applications.
- Typically, a RIA will perform processing on the client, but also communicate with other networked services.
- For example, to persist data in a database.

### **3.Design to take advantage of client processing power.**

RIAs run on the client computer and can take advantage of all the processing power available there.

Consider moving as much functionality as possible onto the client to improve user experience.

- Sensitive business rules should still be executed on the server, because the client logic can be circumvented.

### **4.Design for execution within the browser sandbox**

- RIA implementations have higher security by default and therefore may not have access to all resources on a machine, such as cameras and hardware video acceleration.
- Access to the local file system is limited.
- Local storage is available, but there is a maximum limit.

#### **5. Determine the complexity of your UI requirements.**

- Consider the complexity of your UI. RIA implementations work best when using a single screen for all operations.
- They can be extended to multiple screens, but this requires extra code and screen flow consideration.
- Users should be able to easily navigate or pause, and return to the appropriate point in a screen flow, without restarting the whole process.
- For multi-page UIs, use deep linking methods. Also, manipulate the Uniform Resource Locator (URL), the history list, and the browser's back and forward buttons to avoid confusion as users navigate between screens.

#### **6. Use scenarios to increase application performance or responsiveness.**

- List and examine the common application scenarios to decide how to divide and load components of your application, as well as how to cache data or move business logic to the client.
- To reduce the download and startup time for the application, segregate functionality into separate downloadable components.

#### **7. Design for scenarios where the plug-in is not installed.**

- Because RIA implementations require a browser plug-in, you should design for non-interruptive plug-in installation.
- Consider whether your clients have access to, have permission to, and will want to install the plug-in.
- Consider what control you have over the installation process.

#### **8. Plan for the scenario where users cannot install the plug-in by displaying an informative error message, or by providing an alternative Web UI.**

### **RIA patterns**

- The interaction is the trigger of the RIA pattern.
- Every pattern starts with a user event or a system event, e.g. mouseover, on focus, keyboard stroke or time event.
- A variety of operations can be triggered by the interaction, such as validate, search and refresh.
- Finally, the result of the operation implies an update of the user interface.



## **RIA pattern – Autocompletion**

- User has to deal with a lot of forms and input fields.
- There are forms for analysis parameter, search masks or user registration.
- Unnecessary work would be burdened to the user if he had to fill in every input field.
- Therefore the application should fill in redundant input fields automatically.
- All data captured by application logics could only be suggestions in order to allow a better usability.
- The user is still the last instance for checking the correctness of the suggested data.
- Therefore he should be able to overwrite every auto-completed input field.
- Problem - How could the user of a RIA get an immediate suggestion for values in an input field, after he has entered some initial data or has filled in other related fields?
- Motivation - Assistance by filling in input fields and forms.

(1) People make mistakes when they type.

(2) Typing in data is a tedious work.

(3) Typing speed remains a bottleneck; faster user input is aimed by reducing the number of keystrokes.

(4) Solution - Suggest words or phrases that are likely to complete what the user is typing.

(5) As soon as the user moves to another input element or even as soon as the user inputs a character, the RIA in the background will try to query databases and find relations to already entered data.

(6) If such data could be found and the user has not yet completed it himself, a completed value for the input field is suggested to user.

## **RIA patterns – autosave**

### Use When

- composing email messages
- composing web documents (spreadsheet, text)

### Possible Pitfalls

- if save takes a while can be disruptive to typing
- can cause interface change (as in gmail) leading to unexpected behavior

### Best Practice

- make auto-save as transparent as possible
- email, make it default behavior

- documents, allow it to be turned on
- catch navigation away from page and offer to save

### **RIA pattern- Busy indicator**

#### Use When

- Need to show that system is processing
- want to show indication in context

#### Possible Pitfalls

- can be distracting if not necessary

#### Best Practices

- Place the busy indication as close to the user input as possible
  - Use small animated indicator beside input or inside input field
- Place the busy indication at the place where the results will appear
- Don't use too many indicators as it will make for a noisy interface
- Avoid using indicator if delay is really short

### **RIA pattern – Item selection**

pattern. item selection.

#### Use When

- within a paging context
- need a simple way to provide discontinuous selection

#### Potential Pitfalls

- confusion between checkbox and clicking in row
- mixing with drag and drop
- handling actions on no selection

#### Best Practices

- use only within context of paging; not for scrolled content
- combine with a row of buttons or toolbar that operates on the selected items
- use light shading to re-enforce selected state
- avoid using with drag and drop

### **RIA pattern – live search**

#### Use When

- user needs to search for content and are uncertain on the correct keywords.

#### Potential Pitfalls

- if results are returned too quick, will be distracting
- if results are not returned quick enough, it will feel sluggish

#### Best Practices

- start returning results when the user “slows down” typing
- show results below text entry field for feedback

### **RIA pattern – Refining search**

#### Use When

- user needs to refine a search
- for merchandise search

#### Potential Pitfalls

- sluggish performance

#### Best Practices

- place refining criteria to left of results
- use checkboxes for toggling filters
- use sliders for value ranges
- generally avoid sliders for single values (can combine slider & input)
- provide a “show all” to undo refinement
- try to keep criteria above the fold

### **RESTful Web Services with Nodejs**

- REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.
- Implementation of a REST Web service follows four basic design principles:
- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON), or both.

#### **Use HTTP methods explicitly**

- This REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods.
- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.
- A stateful service keeps track of where the application leaves off while navigating the set. In this stateful design, the service increments and stores a previous Page variable somewhere to be able to respond to requests for next.
- In a Java Platform, Enterprise Edition (Java EE) environment stateful services require a lot of up-front consideration to efficiently store and enable the synchronization of session data across a cluster of Java EE containers
- Stateless server-side components, on the other hand, are less complicated to design, write, and distribute across load-balanced servers.
- A stateless service not only performs better, it shifts most of the responsibility of maintaining state to the client application.

### **Expose directory structure-like URIs**

- This type of URI is hierarchical, rooted at a single path, and branching from it are sub paths that expose the service's main areas.
- A URI is not merely a slash-delimited string, but rather a tree with subordinate and superordinate branches connected at nodes.
- For example, in a discussion threading service that gathers topics ranging from Java to paper, you might define a structured set of URIs like this:
- `http://www.myservice.org/discussion/topics/{topic}`
- The root, /discussion, has a /topics node beneath it. Underneath that there are a series of topic names, such as gossip, technology, and so on, each of which points to a discussion thread. Within this structure, it's easy to pull up discussion threads just by typing something after /topics/.

### **Transfer XML, JSON, or both**

- The objects in your data model are usually related in some way, and the relationships between data model objects (resources) should be reflected in the way they are represented for transfer to a client application.
- In the discussion threading service, an example of connected resource representations might include a root discussion topic and its attributes, and embed links to the responses given to that topic.
- XML representation of a thread

```

<?xml version="1.0"?>
<discussion date="{date}" topic="{topic}">
  <comment>{comment}</comment>
  <replies>
    <reply from="joe@mail.com" href="/discussion/topics/{topic}/joe"/>
    <reply from="bob@mail.com" href="/discussion/topics/{topic}/bob"/>
  </replies>
</discussion>

```

- To give client applications the ability to request a specific content type that's best suited for them, construct your service so that it makes use of the built-in HTTP Accept header, where the value of the header is a MIME type.
- Common MIME types used by RESTful services

### **MIME-Type Content-Type**

- JSON  
application/json
- XML  
application/xml
- XHTML  
application/xhtml+xml
- Node.js is a software system designed for writing highly-scalable internet applications, notably web servers.
- A complete implementation of hello world as an HTTP Server in Node.js:

```

var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(8000);
console.log('Server running at http://localhost:8000/');

```

### **References**

1. Shreeraj Shah, “ Web 2.0 Security: Defending Ajax, RIA, and SOA ”, Course Technology PTR , 2007.
2. Krishna Sankar , Susan A. Bouchard, “ Enterprise Web 2.0 Fundamentals ”, Cisco Press , 2010.



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT – II-Rich Internet Applications – SCS1401**

## **II. Introduction**

**Functions - Objects and Classes - Javascript in Web Browsers - Getting started with Ext JS - Creating your first Ext JS Application - MVC Basics.**

### **Introduction**

Ext JS stands for Extended JavaScript. It is a JavaScript framework and a product of Sencha, based on YUI (Yahoo User Interface). It is basically a desktop application development platform with modern UI. Ext JS is a popular JavaScript framework which provides rich UI for building web applications with cross-browser functionality. Ext JS is basically used for creating desktop applications. It supports all the modern browsers such as IE6+, FF, Chrome, Safari 6+, Opera 12+, etc. Whereas another product of Sencha, Sencha Touch is used for mobile applications.

Ext JS is based on MVC/MVVM architecture. The latest version of Ext JS 6 is a single platform, which can be used for both desktop and mobile application without having different code for different platform.

**MVC** – Model View Controller architecture (version 4)

**MVVM** – Model View Viewmodel (version 5)

### **History**

#### **Ext JS 1.1**

- The first version of Ext JS was developed by Jack Slocum in 2006.
- It was a set of utility classes, which is an extension of YUI.
- He named the library as YUI-ext.

#### **Ext JS 2.0**

- Ext JS version 2.0 was released in 2007.
- This version doesn't have backward compatibility with previous version of Ext JS.

#### **Ext JS 3.0**

- Ext JS version 3.0 was released in 2009.
- This version added new features as chart and list view.
- It had backward compatibility with version 2.0.

#### **Ext JS 4.0**

- Ext JS version 4.0 was released in 2011.
- It had the complete revised structure, which was followed by MVC architecture and a speedy application.

### **Ext JS 5.0**

- Ext JS version 5.0 was released in 2014.
- The major change in this release was to change the MVC architecture to MVVM architecture ability to build desktop apps on touch-enabled devices, two-way data binding, responsive layouts, and many more features.

### **Ext JS 6.0**

- Released in July 1,2015.
- Ext JS 6 merges the Ext JS (for desktop application) and Sencha Touch (for mobile application) framework.

### **Ext JS 7.0**

- Released in August 29,2019.
- The Ext JS 7.0 Modern toolkit supports: drag and drop, row editor and collapsible group for grids, accordion layout, form group, breadcrumb toolbar and drag and drop for tree view.
- Ext JS 7.0 has been upgraded to support Font Awesome 5+, which is SVG based.

### **Features of Ext Js**

- Customizable UI widgets with collection of rich UI such as grids, pivot grids, forms, charts, trees.
- Code compatibility of new versions with the older one.
- A flexible layout manager helps to organize the display of data and content across multiple browsers, devices, and screen sizes.
- Advance data package decouples the UI widgets from the data layer.
- Sophisticated data visualization
- Rich Data Analytics
- Pre integrated and tested UI components
- Layout Manager and responsive configs.
- Ext JS is a client-side JavaScript application framework to build enterprise applications.
- Ext JS supports object-oriented programming concepts using JavaScript which makes easy application development and maintenance.
- Ext JS supports Single Page Application development.



- Ext JS includes MVC and MVVM architecture.
- Ext JS Data package makes it easy to retrieve or saving of data.
- Ext JS includes OOB UI components, containers and layouts.
- Ext JS includes drag and drop functionality for UI containers and components.
- Ext JS includes localization package that makes it easy to localize application.
- Includes OOB themes.

### **Advantages**

- Streamlines cross-platform development across desktops, tablets, and smartphones — for both modern and legacy browsers.
- Increases the productivity of development teams by integrating into enterprise development environments via IDE plugins.
- Reduces the cost of web application development.
- It has set of widgets for making UI powerful and easy.
- It follows MVC architecture so highly readable code.

### **Limitations**

- The size of the library is large, around 500 KB, which makes initial loading time more and makes application slow.
- HTML is full of tags that makes it complex and difficult to debug.
- It is free for open source applications but paid for commercial applications.
- Need quite experienced developer for developing Ext JS applications.

### **Major Browsers support by Ext Js**

- IE 6+
- Mozilla firefox version 1.5+
- Apple safari version 2+
- Opera version 9+
- Chrome 10+

### **Downloading Library Files**

- Download the trial version of Ext JS library files from Sencha <https://www.sencha.com>
- Unzip the folder and you will find various JavaScript and CSS files, which you will include in our application.

- JavaScript Files – JS file which you can find under the folder \ext-6.0.1-trial\ext6.0.1\build are

### JavaScript Files

- **ext.js** -This is the core file which contains all the functionalities to run the application.
- **ext-all.js**-This file contains all the code minified with no comments in the file.
- **ext-all-debug.js**-This is the unminified version of ext-all.js for debugging purpose.
- **ext-all-dev.js**-This file is also unminified and is used for development purpose as it contains all the comments and console logs to check any errors/issue.
- **ext-all.js**-This file is used for production purpose mostly as it is much smaller than any other.

### Difference between Ext.all and Ext.js

- **ext-all.js:** This file contains the entire Ext JS framework (used for Development & testing)
- **ext.js:** This file contains the minimum Ext JS code (Ext JS base library)- used in Production.

### Naming Convention

- Naming convention is a set of rules to be followed for identifiers is shown in Table 2.1.
- It makes the code more readable and understandable to other programmers as well.
- The second word will start with an uppercase letter always.

Name	Convention
Class Name	It should start with an uppercase letter, followed by camel case. For example, StudentClass
Method Name	It should start with a lowercase letter, followed by camel case. For example, doLayout()
Variable Name	It should start with a lowercase letter, followed by camel case. For example, firstName
Constant Name	It should be in uppercase only. For example, COUNT, MAX_VALUE
Property Name	It should start with a lowercase letter, followed by camel case. For example, enableColumnResize = true

**Table 2.1. Naming Convention**

## Architecture

```
-----src
-----resources
-----CSS files
-----Images
-----JavaScript
-----App Folder
-----Controller
-----Controller.js
-----Model
-----Model.js
-----Store
-----Store.js
-----View
-----View.js
-----Utils
-----Utils.js
-----app.js
-----HTML files
```

### Ext Js App

- Ext JS app folder will reside in JavaScript folder of your project.
- The App will contain controller, view, model, store, and utility files with app.js.
- The App will contain **controller, view, model, store, and utility files** with app.js.
- **app.js** – The main file from where the flow of program will start, which should be included in the main HTML file using <script> tag.

### Controller.js

- It is the controller file of Ext JS MVC architecture.
- This contains all the control of the application, the events listeners, and most of the functionality of the code.

- It has the path defined for all the other files used in that application such as store, view, model, require, mixins.

### **View.js**

- It contains the interface part of the application, which shows up to the user.
- Ext JS uses various UI rich views, which can be extended and customized here according to the requirement.

### **Model.js**

- It contains the objects which binds the store data to view.
- It has the mapping of backend data objects to the view dataIndex.
- The data is fetched with the help of store.

### **Store.js**

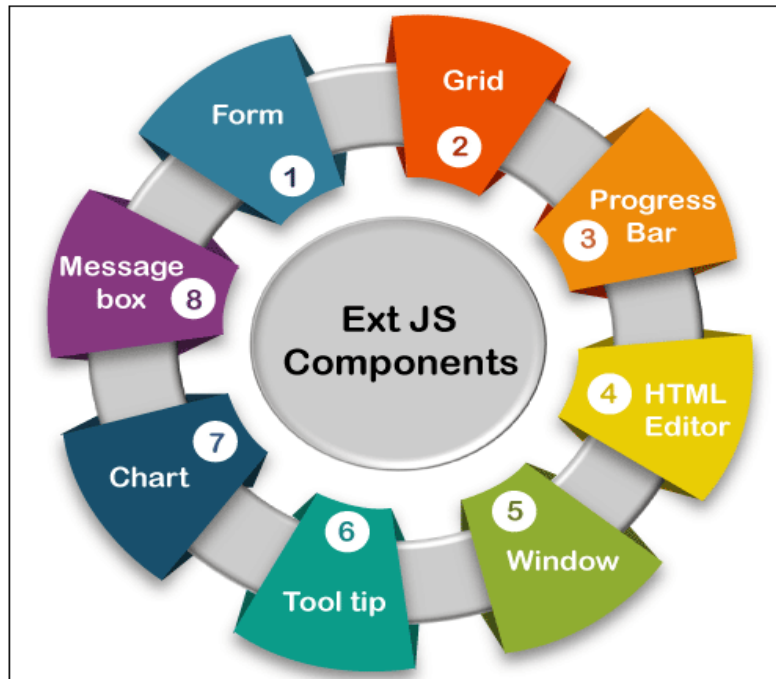
- It contains the data locally cached which is to be rendered on the view with the help of model objects.
- Store fetches the data using proxies which has the path defined for services to fetch the backend data.

### **Utils.js**

- It is not included in MVC architecture but a best practice to use to make the code clean, less complex, and more readable.
- In MVVM architecture, the controller is replaced by ViewModel.

### **Components of Ext JS**

- The components can be defined as **the number of widgets, which helps to create an application.**
- The Component is referred to as the DOM element, which contains the complicated functionality.
- All components are inherited from the **Ext. Component** class.
- Ext JS application includes various types of elements, such as Combobox, grid, panel, container, textfield, numberfield, etc is shown in Figure.2.1.



**Figure.2.1 Ext.Js Components**

The different types of Components and description are shown in Table.2.2.

Sr. No.	Component	Description
1.	<b>Form</b>	The form helps us to obtain the data from the user.
2.	<b>Grid</b>	It is used to display the data in the form of a table.
3.	<b>Chart</b>	It is used to display the data in a pictorial manner.
4.	<b>Message Box</b>	This component is used to display the data in an alert box form.
5.	<b>Window</b>	The <b>Window</b> component helps us to create the window. It is always popped up when an event occurs.
6.	<b>Tool tip</b>	It is used to display the data during the event occurring.

<b>7.</b>	<b>HTML editor</b>	The editor is used to style the data input by the user. It should be color, size, and font.
<b>8.</b>	<b>Progress Bar</b>	This bar always shows the progress of the backend functions.

**Table 2.2 Different types of Components**

### **Alert boxes**

Different type of alert boxes in Ext JS are

- Ext.MessageBox.alert();
- Ext.MessageBox.confirm();
- Ext.MessageBox.wait();
- Ext.MessageBox.prompt();
- Ext.MessageBox.show();

### **Getting Started with Ext Js Applications**

```

<!DOCTYPE html>

<html>

<head>

<link href = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/classic/theme-
classic/resources/theme-classic-all.css"
rel = "stylesheet" />

<script type = "text/javascript"
src = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/ext-all.js"></script> <script
type = "text/javascript">

</script>

</head>

<body>

<div id="helloWorldPanel"></div>

</body>

</html>

```

```

Ext.onReady(function() {
  Ext.create('Ext.Panel', {
    renderTo: 'helloWorldPanel',
    height: 500,
    width: 500,
    title: 'Hello world',
    html: 'Welcome to the world of Ext'
  });
});

```

## Classes in Ext Js

- Ext JS is a JavaScript framework having functionalities of object-oriented programming.
- Ext provides more than 300 classes, which we can use for various functionalities.
- Ext.define() is used for defining the classes in Ext JS.
- An Ext class includes approx. 59 properties and 78 methods.
- Some of the essential methods are **Ext.apply**, **Ext.create**, **Ext.define**, **Ext.getCmp**, **Ext.override**, and **Ext.application** etc.

- **Syntax**

**Ext.define(class name, class members/properties, callback function);**

- **Class Name:** It is the class name that is given by the user depending upon the application structure.
- **Class Member/Properties:** Class member/properties are used to determine the class behavior.
- **Callback Function:** It is a function that is invoked when the class is loaded. It is an optional function to use.

## **Ext.Base**

- Ext.Base is root of all classes created with Ext.define.
- All the classes in Ext JS inherit from Ext.Base

```
Ext.define('Student',
{
    name : 'unnamed',
    getName : function(){
        return "Student name is" + this.name;
    }
}, function(){
    alert('Student object created');
});
```

### Create an Object of a Class

- JavaScript allows us to create an object using new keyword.
- Sencha recommends to use Ext.create() method to create an object of a class which is created using Ext.define() method.

```
var studentObj = Ext.create('Student');
studentObj.getName();
```

### Example

```
<!DOCTYPE html>
<html>
<head>
<link href = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/classic/theme-
classic/resources/theme-classic-all.css"
rel = "stylesheet" />
<script type = "text/javascript"
src = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/ext-all.js"></script>
<script type = "text/javascript">
    Ext.define('Student',
    {
        name : 'suji',
        getName : function(){
```



```

        return "Student name is" + this.name;
    }
}, function(){
    alert('Student object created');
    var studentObj = Ext.create('Student');
    alert(studentObj.getName());
});
</script>
</head>
<body>
<div id="helloWorldPanel"></div>
</body>
</html>

```

## Define Constructor

- A **constructor** is a special function that gets called when a Class is instantiated.

### Example

```

Ext.define('Student', {
    name : 'unnamed',
    getName : function(){
        return 'Student name is ' + this.name;
    },
    constructor : function(studentName){
        if(studentName)
            this.name = studentName;
    }
});

```

- Now you can create an object by passing parameter to the constructor:
- `var studentObj = Ext.create('Student','XYZ');`
- `studentObj.getName();`

## Declare Private Members in Class

- In JavaScript, there are two types of object fields.
- **Public:**
  - accessible from anywhere.
  - They comprise the external interface.
  - Until now we were only using public properties and methods.
- **Private:**
  - accessible only from inside the class.
  - These are for the internal interface.

### Example

```
Ext.define('Student', function(){
    var name = 'abc';
    return {
        constructor : function(name){
            this.name = name;
        },
        getName : function(){
            alert('Student name is' + this.name);
        }
    };
});

//create an object of Student class
var studentObj = Ext.create('Student','XYZ');
studentObj.getName();
```

## Declare Static Members in Class

- The static members in an Ext JS class can be accessed without creating an object using Ext.create() method.

- It can be accessed using class name with dot notation same as other programming languages.
- Declare static members in Ext JS class using 'statics' parameter

### Example

```
Ext.define('Student',
{
    name : 'abc',
    getName : function(){
        alert('Student name is ' + this.name);
    },
    constructor : function(studentName){
        if(studentName)
            this.name = studentName;
    },
    statics :
    {
        getSchoolName : function(){
            return "XYZ";
        }
    }
});

//call static method
alert(Student.getSchoolName());
```

### Inheritance in Ext JS

- Ext JS supports object-oriented programming concepts such as class, inheritance, polymorphism etc.
- We can inherit a new class from existing class using extend keyword while defining a new class in Ext JS.

### Example

```
Ext.define('Person',
{
    name : 'abc',
    getName : function(){
        alert("My name is " + this.name);
    },
    constructor : function(name){
        if(name){
            this.name = name;
        }
    }
});
```

### Example

```
Ext.define('Student',
{
    extend : 'Person',
    schoolName : 'sathyabama',
    constructor : function(name, schoolName){
        this.schoolName = schoolName;
        //call parent class constructor
        this.callParent(arguments);
    },
    getSchoolName : function(){
        alert("My school name is " + this.schoolName);
    }
});

var newStudent = new Student('Queen', 'American School');
newStudent.getName();
newStudent.getSchoolName();
```

## Mixins

- Mixins introduced since Ext JS 4.
- Mixins allows us to use functions of one class as a function of another class without inheritance.
- Mixins can be defined using mixins keyword
- The value of a property will be name of the class where the method is defined.

## Example

```
Ext.define('Person', {
    name: 'abc',
    constructor: function(name) {
        if (name) {
            this.name = name;
        }
    },
    getName: function() {
        alert("My name is " + this.name);
    },
    eat: function(foodType) {
        alert("I like " + foodType);
    }
});

Ext.define('Student', {
    schoolName: 'sathyabama',
    constructor: function(schoolName) {
        this.schoolName = schoolName
    },
    mixins: {
        eat: 'Person'
    },
    },
```

```

        getSchoolName: function() {
            alert("I am a student of " + this.schoolName);
        }
    });
    var studentObj = new Ext.create('Student', 'XYZ');
    studentObj.eat('Sandwich');
    alert(studentObj.getSchoolName());

```

## Config

- Ext JS has a feature called config.
- The config allows you to declare public properties with default values which will be completely encapsulated from other class members.
- Properties which are declared via config, will have get() and set().
- The config properties can be defined using config keyword.
- Initconfig- This will create get and set methods for all the config properties.
  - in the constructor in order to initialize getters and setters.
- The get method returns a value of a config property.
- Set method is used to assign a new value to a config property.
- The name of the **get method** will start from get and suffix with property name
  - get<config property name>()
- **Set method** will be named as
  - set<config property name>().
- The suffix in get and set method names will start from capital character.

## Example

```

Ext.define('Student', {
    config :
    {
        name : 'xxxx',
        schoolName : 'abc'
    },
    constructor : function(config){

```

```

        this.initConfig(config);
    }
});

var newStudent = Ext.create('Student', { name: 'XYZ', schoolName:
'ABC School' });

newStudent.getName();//output: XYZ

newStudent.getSchoolName();//output: ABC School

newStudent.setName('John');

newStudent.setSchoolName('New School');

alert(newStudent.getName());//output: John

alert(newStudent.getSchoolName());

```

### Description

- getName(), setName() and getSchoolName(), setSchoolName() methods in Student object.
- It was automatically created for config properties by Ext JS API.
- Cannot assign config property value directly same as normal class property.

```

newStudent.name = 'bnn'; //Not valid.

newStudent.setName('bvbb');//Valid

```

### Custom Setters

- get and set methods are created for config property automatically.
- Custom setters allow you to add extra logic.
- There are two custom setters: **apply and update**.
- The **apply()** method for config property allows us to add some extra logic before it assigns the value to config property.
- The name of apply method must start with apply with config property name as suffix in CamelCase.

Example: applyName

### Custom Setters -Update()

- The update() method executes after the configuration property value has been assigned.

- The name of update method must start with update with config property name as suffix in CamelCase.
- It has two parameters, newValue and oldValue.

**Example:** updateName : function(newValue, oldValue)

### Example

```
Ext.define('Student',{
    config :
    {
        name : 'unnamed',
        schoolName : 'Unknown'
    },
    constructor : function(config){
        this.initConfig(config);
    },
    applyName : function(name){
        return Ext.String.capitalize(name);
    },
    updateName : function(newValue, oldValue){
        alert('New value: ' + newValue + ', Old value: ' + oldValue);
    }
});

var newStudent = Ext.create('Student', {name : 'xyz', schoolName : 'ABC School'});

newStudent.setName('john');
```

### Alternate ClassName

- Defines alternate names for the class.

### Example

```
Ext.define('Developer',
{
```



```

    alternateClassName: ['Cc'],
    code: function(msg) {
        alert('Typing... ' + msg);
    }
});
var obj = Ext.create('Developer');
obj.code('hi');
var rms = Ext.create('Cc');
rms.code('hello');

```

## requires

- List of classes that have to be loaded when a class is invoked.
- **Example**

```

define('Mother', {
    requires: ['Child'],
    giveBirth: function() {
        // we can be sure that child class is available.
        return new Child();
    }
});

```

## Uses

- List of optional classes to load together with this class.
- **Example**

```

Ext.define('Mother', {
    uses: ['Child'],

```

```

giveBirth: function() {
    // This code might, or might not work:
    // return new Child();

    // Instead use Ext.create() to load the class at the spot if not loaded already:
    return Ext.create('Child');
}

});

```

### **Ext.ClassManager**

- Ext.ClassManager manages all classes and handles mapping from string class name to actual class objects throughout the whole framework.
  - Ext.define
  - create
  - Ext.widget
  - Ext.getClass
  - Ext.getClassName

### **Methods**

- **Get**-Retrieve a class by its name.
- **getByAlias**- Get a reference to the class by its alias.
- **getConfig**-Get a component class name from a config object.
- **getClass** ( object )-Get the class of the provided object.
- **getDisplayName** ( object )-Returns the displayName property or className or object.
- **getName** ( object )-Get the name of the class by its reference or its instance.
- **isCreated** ( className )
  - Checks if a class has already been created.

### **Object**

- **defineProperties** ( obj, props )-Defines new or modifies existing properties directly on an object, returning the object.
- **freeze** ( obj ) -Nothing can be added to or removed from the properties set of a frozen object. Any attempt to do so will fail
- **getOwnPropertyNames** ( obj )-Returns an array of all properties

- **isFrozen** ( obj )-Determines if an object is frozen.
- **isSealed** ( obj )-Determines if an object is sealed.
- **seal** ( obj )-Seals an object, preventing new properties from being added to it and marking all existing properties as non-config
- **preventExtensions** ( obj )-Prevents new properties from ever being added to an object

### What is the difference between freeze and seal in JavaScript ?

- **Object.seal()** allows changes to the existing properties of an object whereas **Object.freeze()** does not allow so.
- **Object.freeze()** makes an object immune to everything even little changes cannot be made.
- **Object.seal()** prevents from deletion of existing properties but cannot prevent them from external changes.
- **Important classes:** The different classes are discussed in Table 2.3.

Class	Description
Ext	The Ext namespace (global object) encapsulates all classes, singletons, and utility methods provided by Sencha libraries.
Ext.Base	The root of all classes created with Ext.define. Ext.Base is the building block of all Ext classes. All classes in Ext inherit from Ext.Base. All prototype and static members of this class are inherited by all other classes.
Ext.ClassManager	Ext.ClassManager manages all classes and handles mapping from string class name to actual class objects throughout the whole framework.
Ext.Loader	Ext.Loader is the heart of the new dynamic dependency loading capability in Ext JS 4+. It is most commonly used via the Ext.require shorthand.

**Table 2.3 Important Classes**

### **Ext.is Class**

- This class checks the platform you are using, whether it is a phone or a desktop, a mac or Windows operating system.
- **Ext.is.Platforms**-This function returns the platform available for this version.
- **Ext.is.Android**-This function will return true, if you are using Android operating system, else it returns false
- **Ext.is.Desktop**-This function will return true, if you are using a desktop for the application, else it returns false
- **Ext.is.Phone**-This function will return true, if you are using a mobile, else it returns false.
- **Ext.is.iPhone**-This function will return true if you are using iPhone, else it returns false.
- **Ext.is.iPod**-This function will return true, if you are using iPod, else it returns false.
- **Ext.is.iPad**-This function will return true, if you are using an iPad, else it returns false.
- **Ext.is.Windows**-This function will return true, if you are using Windows operating system, else it returns false.

### **Ext.supports Class**

- This class provides information if the feature is supported by the current environment of the browser/device or not.
- **Ext.supports.History**- It checks if the device supports HTML 5 history as window.history or not.
- **Ext.supports.GeoLocation**-It checks if the device supports geolocation method or not.
- **Ext.supports.Svg**-It checks if the device supports HTML 5 feature scalable vector graphics (svg) method or not.

### **Ext.String Class**

- Ext.String class has various methods to work with string data. The most used methods are encoding decoding, trim, toggle, urlAppend, etc.
- **Ext.String.trim**-This function is to trim the unwanted space in the string.
- **Ext.String.urlAppend**-This method is used to append a value in the URL string.

### **String**

- It is a global object that may be used to construct String instances.

- String objects may be created by calling the constructor `new String()`.
- **charAt** ( index )-Returns the character at the specified index.
- **concat** ( strings )-Combines combines the text from one or more strings and returns a new string.

```
var hello = "Hello, ";
alert(hello.concat("Kevin", " have a nice day."));
```

Functions

- **Ext.each**
  - Ext.each applies a function to each member of an array.
  - It's basically a more convenient form of a for loop.

#### Example

```
var countries = ['Vietnam', 'Singapore', 'United States', 'Russia'];
Ext.Array.each(countries, function(name, index, countriesItself) {
    alert(countries[index])
});
```

#### Ext.iterate

- Ext.iterate is like Ext.each for non-array objects.
- **Example**

```
ships= { 'Bill': 'wonderful', 'Laura': 'great' };
Ext.iterate(ships, function(key, value) {
    alert(key + "'s ship is the " + value);
});
```

#### Ext.pluck

- Ext.pluck grabs the specified property from an array of objects:

#### Example:

```
var animals = [
    { name: 'Ed', species: 'Unknown' },
    { name: 'Bumble', species: 'Cat' },
```

```

    { name: 'Triumph', species: 'Insult Dog' }
  ];

  alert(Ext.pluck(animals, 'species'));
  alert(Ext.pluck(animals, 'name'));

```

### **Ext.invoke**

- Invoke allows a function to be applied to all members of an array, and returns the results.
- **Example:**

```

var animal=[
    { name: 'Ed', species: 'Unknown'},
    { name: 'Bumble', species: 'Cat'},
    { name: 'Triumph', species: 'Insult Dog' }
];

var describeAnimal = function(animal) {
    return String.format("{0} is a {1}", animal.name, animal.species);
}

var describedAnimals = Ext.invoke(animals, describeAnimal);

alert(describedAnimals); // ['Ed is a Unknown', 'Bumble is a Cat', 'Triumph
is a Insult Dog'];

```

### **Ext.partition**

- Ext.Partition splits an array into two sets based on a function you provide:
- **Example**

```

var trees = [
    { name: 'Oak', height: 20 },
    { name: 'Willow', height: 10 },
    { name: 'Cactus', height: 5 } ];

var isTall = function(tree) { return tree.height > 15 };

alert(Ext.partition(trees, isTall));

```

## Math functions

- **Example**

```
– var numbers = [1, 2, 3, 4, 5];  
– alert(Ext.min(numbers)); //1  
– alert(Ext.max(numbers)); //5  
– alert(Ext.sum(numbers)); //15  
– alert(Ext.mean(numbers)); //3
```

## Date

- The following example converts the date object to a numerical value using number as a function .

- **Example**

```
d = new Date("December 17, 1995 03:24:00");  
alert(Number(d));
```

## Array

- An array is a JavaScript object.
- **Creating an Array**

```
var msgArray = new Array();
```

- **Example**

```
var msgArray = new Array();  
msgArray[0] = "Hello";  
msgArray[99] = "world";  
if (msgArray.length == 100)  
    alert("The length is 100.");
```

## Accessing array elements

- Array elements are nothing less than object properties, so they are accessed as such.
- **Method 1:**

```
var myArray = new Array("Wind", "Rain", "Fire");  
alert(myArray[0]); // "Wind"
```

- **Method 2:**

```
myArray[02]
```

## Methods of Array

- **indexOf** ( searchElement, [fromIndex] )
  - Returns the first index at which a given element can be found in the array, or -1 if it is not present.

### Example

```
var array = [2, 5, 9];  
var index = array.indexOf(2);  
alert(index); // index is 0  
index = array.indexOf(7);  
// index is -1
```

- **join** ( separator )-Joins all elements of an array into a string.

- **Example**

```
var a = new Array("Wind","Rain","Fire");  
var myVar1 = a.join();  
var myVar2 = a.join(", ");  
var myVar3 = a.join(" + ");
```

- **lastIndexOf** ( searchElement, [fromIndex] )
  - Returns the last index at which a given element can be found in the array, or -1 if it is not present.

```
» var array = [2, 5, 9, 2];  
» var index = array.lastIndexOf(2);  
» // index is 3
```

- **pop** -The pop method removes the last element from an array and returns that value to the caller.

```
var myObj = ['angel', 'sonal', 'manal', 'somu'];  
var popped = myObj.pop();  
alert(popped); // Alerts 'somu'
```



- **push** (elements )-Adds one or more elements to the end of an array and returns the new length of the array.

```
var sports = ['soccer', 'baseball'];
sports.push(['football', 'swimming']);
alert(sports);
```

- **Reverse**-Reverses the order of the elements of an array

```
var myArray = ["one", "two", "three"];
alert(myArray.reverse());
```

- **sort**-Sorts the elements of an array.

```
var numbers = [4, 2, 5, 1, 3];
alert(numbers.sort());
```

### Methods of String

- replace ( pattern, replacement )
- search ( regexp )
- toLocaleLowerCase()
- var upperText="SENCHA";
- document.write(upperText.toLocaleLowerCase());
- toLocaleUpperCase
- Trim()
- To UpperCase()

### Ext.Window

- Ext JS take configuration parameters, many of which can be changed at runtime.
- Ext.Window , has a ‘title’ configuration, which takes the default value of ‘Window Title’.
- 4 methods for free – getTitle, setTitle, resetTitle and applyTitle.
- **getTitle** – returns the current title
- **setTitle** – sets the title to a new value
- **resetTitle** – reverts the title to its default value (‘Window Title’)
- **applyTitle** – this is a template method that you can choose to define. It is called whenever setTitle is called.

```
Ext.define('Ext.Window', {
```

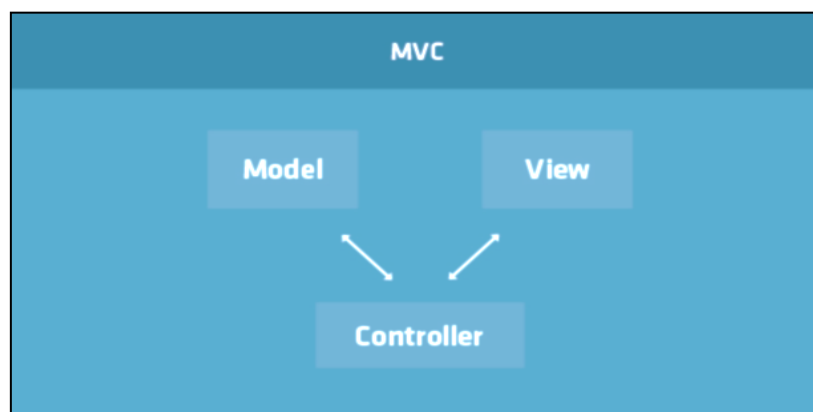
```
//..as above,
config: {
  title: 'Window Title'
},
applyTitle: function(newTitle) {
  this.titleEl.update(newTitle);
}
});
```

### Ext JS MVC framework

- **Model:** It is the collection of fields.
- **Store:** It is the collection of data. Each store is associated with a model. ...
- **View:** It is any type of UI component like grid, chart etc.
- **Controllers:** In controllers we put all the codes that makes our app work.

### Model-View-Controller

- Model-View-Controller (MVC) is an architectural pattern for writing software.
- It divides the user interface of an application into three distinct parts, helping to organize the codebase into logical representations of information depending upon their function shown in Figure 2.2.



**Figure 2.2 MVC Architecture Diagram**

- The *Model* describes a common format for the data being used in the application. It may also contain business rules, validation logic, and various other functions.

- The *View* represents the data to the user. Multiple views may display the same data in different ways (e.g. charts versus grids).
- The *Controller* is the central piece of an MVC application. It listens for events in the application and delegates commands between the *Model* and the *View*.

### **Advantages of MVC**

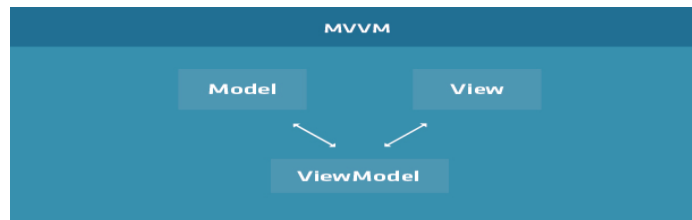
- Easy code maintenance , easy to extend
- MVC Model component can be tested separately from the user
- Easier support for new type of clients
- Development of the various components can be performed parallelly.
- It helps you to avoid complexity by dividing an application into the three units. Model, view, and controller
- It only uses a Front Controller pattern which process web application requests through a single controller.

### **Disadvantages of using MVC**

- Difficult to read, change, to unit test, and reuse this model
- No formal validation support
- Increased complexity
- The difficulty of using MVC with the modern user interface
- There is a need for multiple programmers to conduct parallel programming.
- Knowledge of multiple technologies is required.
- Maintenance of lots of codes in Controller

### **Model-View-ViewModel**

- Model-View-ViewModel (MVVM) is another architectural pattern for writing software that is largely based on the MVC pattern.
- The key difference between MVC and MVVM is that MVVM features an abstraction of a *View* (the *ViewModel*) which manages the changes between a *Model*'s data and the *View*'s representation of that data (i.e. data bindings) — something which typically is cumbersome to manage in traditional MVC applications.



**Figure 2.3 MVVM Architecture Diagram**

### **Elements of the MVVM**

- The Model describes a common format for the data being used in the application, just as in the classic MVC pattern shown in Figure 2.3.
- The View represents the data to the user, just as in the classic MVC pattern.
- The ViewModel is an abstraction of the view that mediates changes between the View and an associated Model. In the MVC pattern, this would have been the responsibility of a specialized Controller, but in MVVM, the ViewModel directly manages the data bindings and formulas used by the View in question.

### **Javascript in web browser**

- **Internet Explorer**
  - On web browser menu click "Tools" menu and select "Internet Options".
  - In the "Internet Options" window select the "Security" tab.
  - On the "Security" tab click on the "Custom level..." button.
  - When the "Security Settings - Internet Zone" dialog window opens, look for the "Scripting" section.
  - In the "Active Scripting" item select "Enable".
  - When the "Warning!" window pops out asking "Are you sure you want to change the settings for this zone?" select "Yes".
  - In the "Internet Options" window click on the "OK" button to close it.
  - Click on the "Refresh" button of the web browser to refresh the page.
- **Internet Explorer < 9**
  - On web browser menu click "Tools" and select "Internet Options".
  - In the "Internet Options" window select the "Security" tab.
  - On the "Security" tab click on the "Custom level..." button.
  - When the "Security Settings - Internet Zone" dialog window opens, look for the "Scripting" section.
  - In the "Active Scripting" item select "Enable".

- When the "Warning!" window pops out asking "Are you sure you want to change the settings for this zone?" select "Yes".
  - In the "Internet Options" window click on the "OK" button to close it.
  - Click on the "Refresh" button of the web browser to refresh the page.
- Mozilla Firefox < 4
  - On the web browser menu click "Tools" and select "Options".
  - In the "Options" window select the "Content" tab.
  - Mark the "Enable JavaScript" checkbox.
  - In the opened "Options" window click on the "OK" button to close it.
  - Click on the "Reload current page" button of the web browser to refresh the page.
- Google Chrome
  - On the web browser menu click on the "Customize and control Google Chrome" and select "Options".
  - In the "Google Chrome Options" tab select the "Under the Hood" menu item.
  - In the "Privacy" section click "Content settings..." button.
  - In the "Content settings" window go to "JavaScript" section and select "Allow all sites to run JavaScript (recommended)".
  - Close the "Google Chrome Options" tab.
  - Click on the "Reload this page" button of the web browser to refresh the page.
- Google Chrome < 10
  - On the web browser menu click on the "Customize and control Google Chrome" and select "Options".
  - In the "Google Chrome Options" window select the "Under the Hood" tab.
  - In the "Privacy" section click "Content settings..." button.
  - In the "Content settings" window select the "JavaScript" tab.
  - In the "JavaScript" tab select "Allow all sites to run JavaScript (recommended)".
  - Close the "Content Setting" window.
  - Close the "Google Chrome Options" window.
  - Click on the "Reload this page" button of the web browser to refresh the page.
- Apple Safari
  - On the web browser menu click on the "Edit" and select "Preferences".

- In the "Preferences" window select the "Security" tab.
- In the "Security" tab section "Web content" mark the "Enable JavaScript" checkbox.
- Click on the "Reload the current page" button of the web browser to refresh the page.
- Opera
  - 1. a) Click on "Menu", over mouse on the "Settings" then over mouse on the "Quick preferences" and mark the "Enable JavaScript" checkbox.
  - 1. b) If "Menu bar" is shown click on the "Tools", hover mouse on the "Quick preferences" and mark the "Enable JavaScript" checkbox.
- Opera < v. 10
  - On the web browser menu click "Tools" and select "Preferences".
  - In the "Preferences" window select the "Advanced" tab.
  - On the "Advanced" tab click on "Content" menu item.
  - Mark the "Enable JavaScript" checkbox.
  - In the opened "Preferences" window click on the "OK" button to close it.
  - Click on the "Reload" button of the web browser to refresh the page.

## REFERENCES

1. David Flanagan, "JavaScript: The Definitive Guide", 6th Edition, O'Reilly Media, Inc, March 2011.
2. Alessio Malizia, Ext JS Documentation Site "Mobile 3D Graphics", Springer, 2006.
3. Colin Ramsay, Shea Frederick, Steve Cutter' Blades, "Learning Ext JS", Packt Publishing, 2008.
4. Jesus Garcia, "Ext JS in Action", Manning Publications, 2010.



# **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)**

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **UNIT – III - Rich Internet Applications – SCS1401**

### III. Introduction

#### **Fundamental Classes - Event Handling - Panels and Grids - Layouts and Widgets - Working with data.**

##### **Components in Ext JS**

- UI is made up of one or more widgets called Components.
- Ext JS provides different types of components out of the box with complex functionalities, which you can use in your application such as Viewport, panel, container, grid, textfield, combobox, numberfield etc. .
- Component is a group of DOM elements with complex functionality.
- Ext JS UI components are derived from the Ext.Component class, which in-turn is derived from the Ext.Base class.
- All the Ext JS components inherit from Ext.Component class.
- Ext.Component class is derived from the Ext.Base class.
- All the components in Ext JS are registered with Ext.ComponentManager class on creation.
- It can be accessed by id using Ext.getCmp() method.
- Example: Ext.getCmp('myComponent')

##### **Example**

```
Ext.onReady(function () {  
    Ext.create('Ext.Component', {  
        id:'myComponent',  
        renderTo: Ext.getBody(), //render method to render to body  
        html:'Hello World!'  
    });  
});
```

##### **Component Life Cycle**

- Each component in Ext JS passes through following three stages is shown in Table 3.1.



Stage	Description
Initialization	Registering the component with Ext.ComponentManager and deciding if a component will be rendered.
Rendering	Creating the DOM for the component and adding it to the DOM hierarchy with the events, CSS, etc..
Destruction	Removing the events, the DOM object and unregistering the component from the Ext.ComponentManager

**Table 3.1 Stages of Component Life Cycle**

### Ext JS UI Components

Every Component has a symbolic name called 'xtype'.

Example: Ext.panel.Panel has xtype : 'panel' is discussed in Table 3.2.

Component Name	Class Name	Xtype
ComboBox	Ext.form.field.ComboBox	combobox or combo
Radio Button	Ext.form.field.Radio	radio or radiofield
Checkbox	Ext.form.field.Checkbox	checkbox or checkboxfield
TextField	Ext.form.field.Text	Textfield
Label	Ext.form.Label	Label
Button	Ext.button.Button	Button
DateField	Ext.form.field.Date	Datefield
File Upload	Ext.form.field.File	filefield, fileuploadfield
Hidden Field	Ext.form.field.Hidden	Hidden
Number Field	Ext.form.field.Number	Numberfield
Spinner	Ext.form.field.Spinner	Spinnerfield

Text Area	Ext.form.field.TextArea	Textarea
Time Field	Ext.form.field.Time	Timefield
Trigger	Ext.form.field.Trigger	triggerfield, trigger
Chart	Ext.chart.CartesianChart	Chart
Html Editor	Ext.form.field.HtmlEditor	Htmleditor

**Table 3.2 Ext JS UI Components**

### Combo box

```
var states = Ext.create('Ext.data.Store',
{
    fields: ['abbr', 'name'],
    data : [    {"abbr": "AL", "name": "Sonal"},
    {"abbr": "AK", "name": "Monal"},
    {"abbr": "AZ", "name": "Konal"}    ]
});

Ext.create('Ext.form.ComboBox',
{
    fieldLabel: 'Choose State',
    store: states,
    queryMode: 'local',
    displayField: 'name',
    valueField: 'abbr',
    renderTo: Ext.getBody();
});
```

### Text

- The Text field has a useful set of validations and able to input the data.

#### Example

```
Ext.create('Ext.form.Panel',
{
    title: 'Contact Info',
    width: 300,
```

```

        bodyPadding: 10,
        renderTo: Ext.getBody(),
        items: [{
            xtype: 'textfield',
            name: 'name',
            fieldLabel: 'Name',
            allowBlank: false
        },
        {
            xtype: 'textfield',
            name: 'email',
            fieldLabel: 'Email Address',
            vtype: 'email'
        }
    ]});

```

## Label

```

Ext.create('Ext.form.Panel',
{
    title: 'Field with Label',
    width: 400,
    bodyPadding: 10,
    renderTo: Ext.getBody(),
    layout: { type: 'hbox',
    align: 'middle'
    },
    items: [{
        xtype: 'textfield',
        hideLabel: true,
        flex: 1
    },.]);

```

## Button

```
Ext.create('Ext.Button',
{
    text: 'Click me',
    renderTo: Ext.getBody(),
    handler: function()
    {
        alert('You clicked the button!');
    });
```

## Date

```
Ext.create('Ext.form.Panel',
{
    renderTo: Ext.getBody(),
    width: 300,
    bodyPadding: 10,
    title: 'Dates',
    items: [{
        xtype: 'datefield',
        fieldLabel: 'From',
        name: 'from_date',
        maxValue: new Date()
    }]);
```

## Number

```
Ext.create('Ext.form.Panel',
{
    title: 'Numberings',
    width: 300,
    bodyPadding: 10,
    renderTo: Ext.getBody(),
```

```

items: [{
    xtype: 'numberfield',
    name: 'bottles',
    fieldLabel: 'Number starts',
    value: 99,
    maxValue: 99,
    minValue: 0
}],});

```

### **Text Area**

```

Ext.create('Ext.form.FormPanel',
{
    title    : 'My Document',
    width    : 400,
    bodyPadding: 10,
    renderTo : Ext.getBody(),
    items: [{
        xtype    : 'textareafield',
        name      : 'message',
        fieldLabel: 'Message',
        anchor    : '100%'
    }]);

```

### **HTML Editor**

```

Ext.tip.QuickTipManager.init();
Ext.create('Ext.form.HtmlEditor',
{
    width: 580,
    height: 250,
    renderTo: Ext.getBody()
});

```

### **Charts**

- Charts are used to represent data in pictorial format.
- The different charts provided by Ext JS –
  - Pie Chart
  - Line Chart
  - Bar Chart
  - Area Chart

### Pie Chart

- Syntax

```
Ext.create('Ext.chart.PolarChart', {
    series: [{
        Type: 'pie'
        ..
    }]
    render and other properties.
});
```

### Container

- Ext JS includes components that can contain other components are called **container**.
- Ext.container.Container is a base class for all the containers in Ext JS.
- Add different types of Ext JS components into a container using **items** config property or **add()** method.
- Container can also include another container.
- We can add or remove the components from the container and from its child elements.
- Ext.container.Container is the base class for all the containers in Ext JS.

### Example

```
Ext.create('Ext.container.Container',
{
    layout:
    {
        type: 'hbox'
```

```

    },
    width: 400,
    renderTo: Ext.getBody(),
    border: 1,
    defaults: {
        labelWidth: 80, xtype: 'datefield',
    },
    items: [{
        xtype: 'datefield',
        name: 'Dates',
        fieldLabel: 'Dates'
    }]);

```

### **Components inside container**

```

var component1 = Ext.create('Ext.Component', {
    html: 'First Component'
});

Ext.create('Ext.container.Container', {
    renderTo: Ext.getBody(),
    items: [component1]
});

```

### **Container inside container**

```

var container = Ext.create('Ext.container.Container', {
    items: [component3, component4]
});

Ext.create('Ext.container.Container', {
    renderTo: Ext.getBody(),
    items: [container]
});

```

### **Types of Container**

- Ext.panel.Panel

- Ext.form.Panel
- Ext.tab.Panel
- Ext.container.Viewport

### **Ext.panel.Panel**

- It is the basic container which allows to add items in a normal panel.

```

var childPanel1 = Ext.create('Ext.Panel', {
    html: 'First Panel'
});

Ext.create('Ext.panel.Panel', {
    renderTo: Ext.getBody(),
    width: 100,
    height : 100,
    border : true,
    frame : true,
    items: [ childPanel1]
});

```

### **Ext.form.Panel Container**

- Form panel provides a standard container for forms.
- Ext.panel.Panel, which automatically creates a BasicForm for to manage any Ext.form.field.Field objects.
- **Example**

```

var child1 = Ext.create('Ext.Panel',{
    html: 'Text field'
});

Ext.create('Ext.form.Panel', {
    renderTo: Ext.getBody(),
    width: 100,
    height : 100,
    border : true,

```



```

        frame : true,
        layout : 'auto', // auto is one of the layout type.
        items : [child1]
    });

```

### **Ext.tab.Panel**

- Tab panel is like a normal panel but has support for card tab panel layout.
- **Example**

```

Ext.create('Ext.tab.Panel', {
    renderTo: Ext.getBody(),
    height: 100,
    width: 200,

    items: [{
        xtype: 'panel',
        title: 'Tab One',
        html: 'The first tab',

    }, {
        // xtype for all Component configurations in a Container
        title: 'Tab Two',
        html: 'The second tab',

    }]
});

```

### **Ext.container.Viewport**

- Viewport is a container that automatically resizes itself to the size of the whole browser window.

### **Example**

```
var childPanel1= Ext.create('Ext.panel.Panel', {  
    title: 'Child Panel 1',  
    html: 'Another Panel'  
});  
  
Ext.create('Ext.container.Viewport', {  
    renderTo: Ext.getBody(),  
    items: [childPanel1 ]  
});
```

### **Layout**

- Layout is the way the elements are arranged in a container.
- It can be horizontal, vertical, or any other.
- Ext JS has a different layout defined in its library but we can always write custom layouts

### **Types of Layout**

- Absolute Layout
- Accordion Layout
- Anchor Layout
- Border Layout
- Auto Layout
- Card Layout
- Column Layout
- Fit Layout
- Table Layout
- VBox Layout
- HBox Layout

### **Absolute Layout**

- This layout allows to position the items using XY coordinates in the container.

### **Syntax**

Layout: 'absolute'

### Example

```
Ext.create('Ext.container.Container', {  
    renderTo: Ext.getBody(),  
    layout: 'absolute',  
    items: [{  
        title: 'Panel 1',  
        x: 50,  
        y: 50,  
        html: 'Positioned at x:50, y:50',  
        width: 500,  
        height: 100  
    }]  
});
```

### Accordion Layout

- This layout allows to place all the items in stack fashion (one on top of the other) inside a container.

Syntax:

```
layout: 'accordion'
```

### Example

```
renderTo : Ext.getBody(),  
layout : 'accordion',  
width : 600,  
  
items : [{  
    title : 'Panel 1',  
    html : 'Panel 1 html content'  
},{  
    title : 'Panel 2',  
    html : 'Panel 2 html content'  
},{  
    title : 'Panel 3',
```

```

        html : 'Panel 3 html content'
    }, {
        title : 'Panel 4',
        html : 'Panel 4 html content'
    }, {
        title : 'Panel 5',
        html : 'Panel 5 html content'
    }
  ]
});

```

### Anchor Layout

- This layout gives the privilege to the user to specify the size of each element with respect to the container size.

#### Syntax

```
layout: 'anchor'
```

### Example

```

Ext.create('Ext.container.Container', {
    renderTo : Ext.getBody(),
    layout : 'anchor',
    width : 600,

    items : [{
        title : 'Panel 1',
        html : 'panel 1',
        height : 100,
        anchor : '50%'
    }
  ]
});

```

### Border Layout

- In this layout, various panels are nested and separated by borders.

## Syntax

Layout: 'Border'

## Example

```
Ext.create('Ext.panel.Panel', {  
    renderTo: Ext.getBody(),  
    height: 300,  
    width: 600,  
    layout: 'border',  
  
    defaults: {  
        collapsible: true,  
        split: true,  
        bodyStyle: 'padding:15px'  
    },  
    items: [{  
        title: 'Panel1',  
        region: 'west',  
        html: 'This is Panel 1'  
    }]  
});
```

## Auto Layout

- This is the default layout that decides the layout of the elements based on the number of elements.

## Syntax

layout: 'auto'

## Example

```
Ext.create('Ext.container.Container', {  
    renderTo : Ext.getBody(),  
    layout : 'auto',  
    width : 600,
```

```

items : [{
    title: 'First Component',
    html : 'This is First Component'
},{
    title: 'Second Component',
    html : 'This is Second Component'
},{
    title: 'Third Component',
    html : 'This is Third Component'
},{
    title: 'Fourth Component',
    html : 'This is Fourth Component'
}]
});

```

## Card

- This layout arranges different components in tab fashion.
- Tabs will be displayed on top of the container.
- Every time only one tab is visible and each tab is considered as a different component.

## Example

```

Ext.create('Ext.tab.Panel', {
    renderTo: Ext.getBody(),
    requires: ['Ext.layout.container.Card'],
    xtype: 'layout-cardtabs',
    width: 600,
    height: 200,
    items:[{
        title: 'Tab 1',
        html: 'This is first tab.'
    },{
        title: 'Tab 2',

```

```

        html: 'This is second tab.'
    }, {
        title: 'Tab 3',
        html: 'This is third tab.'
    }]
});

```

## Column

- This layout is to show multiple columns in the container.
- We can define a fixed or percentage width to the columns.
- The percentage width will be calculated based on the full size of the container.

### Example

```

Ext.create('Ext.panel.Panel', {
    renderTo : Ext.getBody(),
    layout : 'column' ,
    xtype: 'layout-column',
    requires: ['Ext.layout.container.Column'],
    width : 600
    items: [{
        title : 'First Component width 30%',
        html : 'This is First Component',
        columnWidth : 0.30
    }, {
        title : 'Second Component width 40%',
        html : '<p> This is Second Component </p> <p> Next line for
second component </p>',
        columnWidth : 0.40
    }]
});

```

## Fit

- In this layout, the container is filled with a single panel.
- When there is no specific requirement related to the layout, then this layout is used.

## Example

```
Ext.create('Ext.container.Container', {  
    renderTo : Ext.getBody(),  
    layout : {  
        type : 'fit'  
    },  
    width : 600,  
    defaults : {  
        bodyPadding: 15  
    },  
    items : [{  
        title: 'Panel1',  
        html : 'This is panel 1'  
    }, {  
        title: 'Panel2',  
        html : 'This is panel 2'  
    }, {  
        title: 'Panel3',  
        html : 'This is panel 3'  
    }, {  
        title: 'Panel4',  
        html : 'This is panel 4'  
    }  
    ]  
});
```

## Table Layout

- As the name implies, this layout arranges the components in a container in the HTML table format.



## Example

```
Ext.create('Ext.container.Container', {
    renderTo : Ext.getBody(),
    layout : {
        type : 'table',
        columns : 3,
        tableAttrs: {
            style: {
                width: '100%'
            }
        }
    },
    width:600,
    height:200,

    items : [{
        title : 'Panel1',
        html : 'This panel has colspan = 2',
        colspan : 2
    }, {
        title : 'Panel2',
        html : 'This panel has rowspan = 2',
        rowspan: 2
    }, {
        title : 'Panel3',
        html : 'This is panel 3'
    }, {
        title : 'Panel4',
        html : 'This is panel 4'
    }, {
```

```

        title : 'Panel5',
        html : 'This is panel 5'
    }]
});

```

## **vBox**

- This layout allows the element to be distributed in the vertical manner. This is one of the mostly used layout.

## **Example**

```

Ext.create('Ext.panel.Panel', {
    renderTo : Ext.getBody(),
    layout : {
        type : 'vbox',
        align: 'stretch'
    },
    requires: ['Ext.layout.container.VBox'],
    xtype: 'layout-vertical-box',
    width : 600,
    height :400,
    frame :true,
    items : [{
        title: 'Panel 1',
        html : 'Panel with flex 1',
        margin: '0 0 10 0',
        flex : 1
    },{
        title: 'Panel 2',
        html : 'Panel with flex 2',
        margin: '0 0 10 0',
        flex : 2
    }]
});

```

```
});
```

## Hbox

- This layout allows the element to be distributed in the horizontal manner.

## Example

```
Ext.create('Ext.panel.Panel', {  
    renderTo : Ext.getBody(),  
    layout : {  
        type : 'hbox'  
    },  
    requires: ['Ext.layout.container.HBox'],  
    xtype: 'layout-horizontal-box',  
    width : 600,  
    frame : true,  
    items : [{  
        title: 'Panel 1',  
        html : 'Panel with flex 1',  
        flex : 1  
    }, {  
        title: 'Panel 2',  
        html : 'Panel with flex 2',  
        flex : 2  
    }, ]  
});
```

## Working with Data

- Data package is used for loading and saving all the data in the application.
- Data package has numerous number of classes but the most important classes are –
- Model
- Store
- Proxy

## Model

- The base class for model is **Ext.data.Model**.
- It represents an entity in an application.
- It binds the store data to view.
- It has mapping of backend data objects to the view dataIndex.
- The data is fetched with the help of store.

### Creating a Model

- For creating a model, we need to extend Ext.data.Model class and we need to define the fields, their name, and mapping.

```
Ext.define('StudentDataModel', {
    extend: 'Ext.data.Model',
    fields: [
        { name: 'name', mapping : 'name' },
        { name: 'age', mapping : 'age' },
        { name: 'marks', mapping : 'marks' }
    ]
});
```

- The name should be the same as the dataIndex, which we declare in the view and the mapping should match the data, either static or dynamic from the database, which is to be fetched using store.

### Store

- The base class for store is **Ext.data.Store**.
- It contains the data locally cached, which is to be rendered on view with the help of model objects. Store fetches the data using proxies, which has the path defined for services to fetch the backend data.
- Store data can be fetched in two ways - static or dynamic.

### Static store

- For static store, we will have all the data present in the store as shown in the following code.

```
Ext.create('Ext.data.Store', {
    model: 'StudentDataModel',
    data: [
        { name : "Asha", age : "16", marks : "90" },
    ]
});
```

```

        { name : "Vinit", age : "18", marks : "95" },
        { name : "Anand", age : "20", marks : "68" },
        { name : "Niharika", age : "21", marks : "86" },
        { name : "Manali", age : "22", marks : "57" }
    ];
});

```

## Dynamic Store

- Dynamic data can be fetched using proxy.
- We can have proxy which can fetch data from Ajax, Rest, and Json.

## Proxy

- The base class for proxy is Ext.data.proxy.Proxy.
- Proxy is used by Models and Stores to handle the loading and saving of Model data.
- There are two types of proxies
- Client Proxy
- Server Proxy

## Types of proxy

- Client Proxy-Client proxies include Memory and Local Storage using HTML5 local storage.
- Server Proxy-Server proxies handle data from the remote server using Ajax, Json data, and Rest service.

## Defining proxies in the server

```

Ext.create('Ext.data.Store', {
    model: 'StudentDataModel',
    proxy : {
        type : 'rest',
        actionMethods : {
            read : 'POST' // Get or Post type based on requirement
        },
        url : 'restUrlPathOrJsonFilePath', // here we have to include the rest URL
        path
        // which fetches data from database or Json file path where the data is stored
    }
});

```

```

        reader: {
            type : 'json', // the type of data which is fetched is of JSON type
            root : 'data'
        },
    }
});

```

## Panels

- A panel is a container that has specific functionality and structural components that make it the perfect building block for application-oriented user interfaces.
- We can add toolbars, buttons at the bottom or make the panel collapsible.
- The panels can be assigned easily to another container.
- **Example**

```

var main = new Ext.Panel({
    title: 'My first panel', //panel's title
    width:250, //panel's width
    height:300, //panel's height
    renderTo: 'frame', //the element where the panel is going to be inserted
    html: 'Nothing important just dummy text' //panel's content
});

```

## Types of Panel

### Tree Panel

- The Tree Panel Component is one of the most versatile Components in Ext JS and is an excellent tool for displaying heirarchical data in an application.
- Tree Panel extends from the same class as Grid Panel, so all of the benefits of Grid Panels - features, extensions, and plugins can also be used on Tree Panels.
- Things like columns, column resizing, dragging and dropping, renderers, sorting and filtering can be expected to work similarly for both components.

### Example

```

Ext.create('Ext.tree.Panel', {
    renderTo: Ext.getBody(),

```

```

    title: 'Simple Tree',
    width: 150,
    height: 150,
    root: {
        text: 'Root',
        expanded: true,
        children: [
            {
                text: 'Child 1',
                leaf: true
            },
            {
                text: 'Child 2',
                leaf: true
            },
            {
                text: 'Child 3',
                expanded: true,
                children: [
                    {
                        text: 'Grandchild',
                        leaf: true
                    }
                ]
            }
        ]
    }
});

```

## Grid Panels

- ExtJS Grid, you can create grids which are nothing but tables with rows and columns in it.

- Based on the type of ExtJS Grid you use, you can print and edit the grid contents.
- Edit includes Add/Modify/Delete of a particular column/row in the table.
- ExtJS Grid also takes care of paging when the number of records is huge.

### Types of Grid Panels in ExtJS Grid

- **Grid Panel** – Using GridPanel you are allowed to show only the data on the Grid. No other transactions can be done on the Grid.
- **Editor Grid Panel** – Editor Grid Panel extends or inherits from GridPanel. If any of the columns in the Editor Grid Panel has to be editable, then you can configure specific editor for that column.
- **Property Grid** – Property Grid extends or inherits from EditorGridPanel. It is used to manage elements of the Grid as key-value pairs.

### Events

- Events are something which get fired when something happens to the class.
- For example, when a button is getting clicked or before/after the element is rendered.

### Methods of Writing Events

- Built-in events using listeners
- Attaching events later
- Custom events

### Built-in Events Using Listeners

```
Ext.create('Ext.Button', {
    renderTo: Ext.getElementById('helloWorldPanel'),
    text: 'My Button',

    listeners: {
        click: function() {
            Ext.MessageBox.alert('Alert box', 'Button is clicked');
        }
    }
});
```



## Attaching an Event Later

- In the previous method of writing events, we have written events in listeners at the time of creating elements.

### Example

```
button.on('click', function() {  
    Ext.MessageBox.alert('Alert box', 'Button is clicked');  
});
```

### Example

```
var button = Ext.create('Ext.Button',  
    {  
        renderTo: Ext.getBody(),  
        text: 'My Button'  
    });  
button.on('click', function()  
{    alert("Success!','Event listener attached by .on');  
});
```

## Custom Events

- We can write custom events in Ext JS and fire the events with fireEvent method.
- FireEvent- Events fire whenever something interesting happens to one of your Classes.
- when a Component renders to the screen, Ext JS fires an event after the render completes.

## Removing Listeners

- we can add listeners at any time, we can also remove them.
- Use .un function to remove the listener.

### Example

```
doSomething = function() {  
    Ext.Msg.alert('listener called');  
};  
var button = Ext.create('Ext.Button', {  
    renderTo: Ext.getBody(),
```

```

    text: 'My Button',
    listeners: {
        click: doSomething,
    }
});
Ext.defer(function() {
    button.un('click', doSomething);
}, 3000);

```

### **Ext.defer**

- Clicking the button in the first 3 seconds yields an alert message.
- However, after 3 seconds the listener is removed so nothing happens

## **REFERENCES**

1. David Flanagan, "JavaScript: The Definitive Guide", 6th Edition, O'Reilly Media, Inc, March 2011.
2. Alessio Malizia, Ext JS Documentation Site "Mobile 3D Graphics", Springer, 2006.
3. Colin Ramsay, Shea Frederick, Steve Cutter, Blades, "Learning Ext JS", Packt Publishing, 2008.
4. Jesus Garcia, "Ext JS in Action", Manning Publications, 2010.



# **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(DEEMED TO BE UNIVERSITY)**

**Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE**

**[www.sathyabama.ac.in](http://www.sathyabama.ac.in)**

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT – IV-Rich Internet Applications – SCS1401**

## **IV.Introduction to GWT**

**Overview - Why Gwt - GWT compiler - Cross browser compatibility. Basic GWT - GWT modules - Build and deploy simple GWT application - GWT Widget Library.**

### **Overview**

Google Web Toolkit (GWT) is a development toolkit to create Rich Internet Applications (RIA).

### **Features of GWT are**

- GWT provides developers option to write client-side application in JAVA.
- GWT compiles the code written in JAVA to JavaScript code.
- Application written in GWT is cross-browser compliant. GWT automatically generates Java script code suitable for each browser.
- GWT is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.
- GWT is a framework to build large-scale high-performance web application while keeping them as easy-to-maintain.
- Browser supports (IE, Firefox, Mozilla, Safari, and Opera)
- Browser history management
- Dynamic, reusable UI components
- Really simple RPC
- Real debugging
- Completely Open Source

### **Why GWT?**

- All of our favourite Java development tools such as Eclipse, Junit, IntelliJ and J Profiler can be used for AJAX development.
- Static type checking in Java language boosts productivity while reducing errors.
- Common Java script errors are easily caught at compile time rather than by users at run time.
- Code prompting/completion is widely available.
- Java based OO designs are easier to communicate and understand, thus making AJAX code base more comprehensible with less documentation.
- GWT provides easy integration with Junit and Maven.
- Being Java based, GWT has a low learning curve for Java Developers.
- GWT generates optimized Java script code, produces browser's specific Javascript code by self.
- GWT provides Widgets library provides most of tasks required in an application.
- GWT is extensible and custom widget can be created to cater to application needs.

- On top of everything, GWT applications can run on all major browsers and smart phones including Android and iOS-based phones/tablets.

### **Disadvantages of GWT**

Although GWT offers plenty of advantages, it suffers from the following disadvantages

- Not Indexable – Web pages generated by GWT would not be indexed by search engines because these applications are generated dynamically.
- Not Degradable – If your application user disables Javascript then user will just see the basic page and nothing more.
- Not Designer's Friendly – GWT is not suitable for web designers who prefer using plain HTML with placeholders for inserting dynamic content at later point in time

### **The GWT Components**

The GWT framework can be divided into following three major parts –

- GWT Java to JavaScript compiler – This is the most important part of GWT which makes it a powerful tool for building RIAs. The GWT compiler is used to translate all the application code written in Java into JavaScript.
- JRE Emulation library – Google Web Toolkit includes a library that emulates a subset of the Java runtime library. The list includes java.lang, java.lang.annotation, java.math, java.io, java.sql, java.util and java.util.logging
- GWT UI building library – This part of GWT consists of many subparts which includes the actual UI components, RPC support, History management, and much more.

GWT also provides a GWT Hosted Web Browser which lets you run and execute your GWT applications in hosted mode, where your code runs as Java in the Java Virtual Machine without compiling to JavaScript.

### **Cross browser compatibility**

GWT shields you from worrying too much about cross-browser incompatibilities. If you stick to built-in widgets and composites, your applications will work similarly on the most recent versions of Internet Explorer, Firefox, Chrome, and Safari. (Opera, too, most of the time.) HTML user interfaces are remarkably quirky, though, so make sure to test your applications thoroughly on every browser.

Whenever possible, GWT defers to browsers' native user interface elements. For example, GWT's Button widget is a true HTML `<button>` rather than a synthetic button-like widget built, say, from a `<div>`. That means that GWT buttons render appropriately in different browsers and on different client operating systems. We like the native browser controls because they're fast, accessible, and most familiar to users.

### **Basic Widgets**

Every user interface considers the following three main aspects –

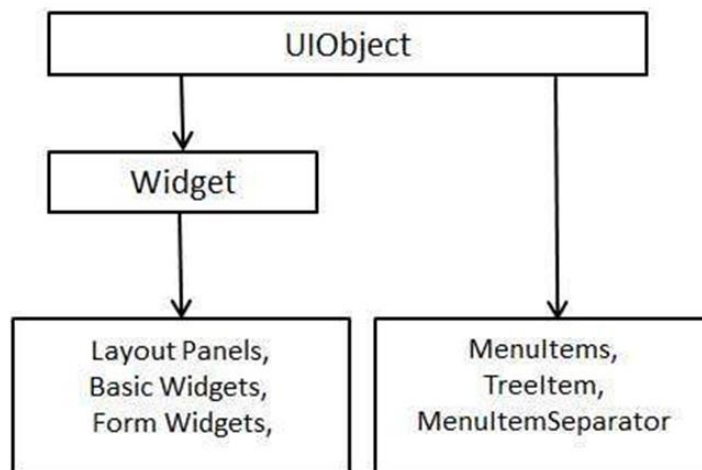
**UI elements** – These are the core visual elements the user eventually sees and interacts with.

**Layouts** – They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).

**Behavior** – These are events which occur when the user interacts with UI elements.

#### GWT UI Elements

The GWT library provides classes in a well-defined class hierarchy to create complex web-based user interfaces. All classes in this component hierarchy has been derived from the UIObject base class shown in Figure 4.1.



**Figure 4.1 Hierarchical structure of GWT UI elements**

Every Basic UI widget inherits properties from Widget class which in turn inherits properties from UIObject.

GWT Basic widgets are:

**Label:** The Label can contain only arbitrary text and it cannot be interpreted as HTML. This widget uses a <div> element, causing it to be displayed with block layout.

**HTML:** This widget can contain HTML text and displays the html content using a <div> element, causing it to be displayed with block layout.

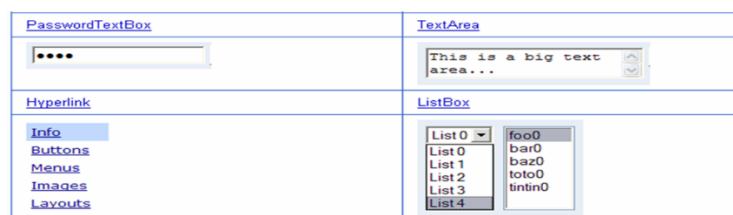
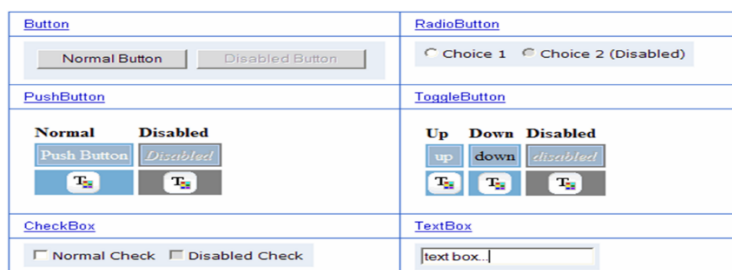
**Image:** This widget displays an image at a given URL.

**Anchor:** This widget represents a simple <a> element.

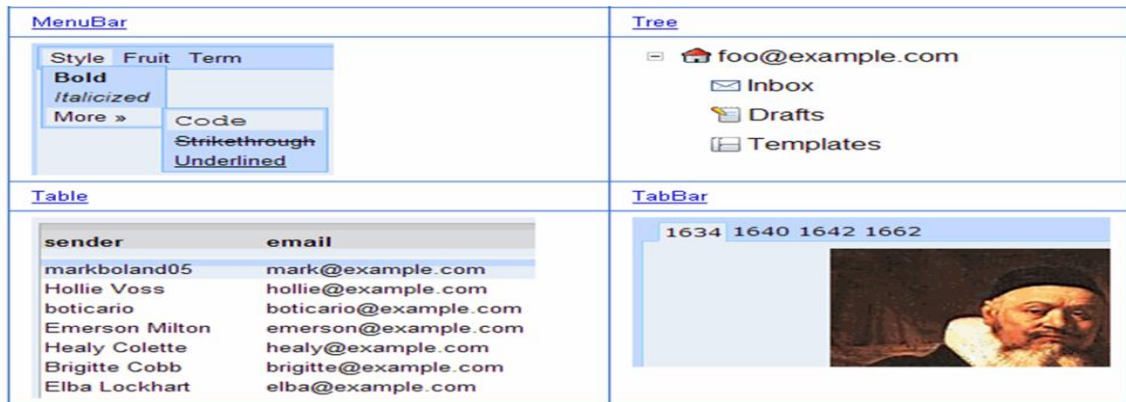
#### **GWT Form widgets:**

The GWT form widgets are discussed in Figure 4.2 and Figure 4.3.

- Button -This widget represents a standard push button.
- PushButton - This widget represents a normal push button with custom styling.
- ToggleButton - This widget represents a stylish stateful button which allows the user to toggle between up and down states.
- CheckBox - This widget represents a standard check box widget. This class also serves as a base class for RadioButton.
- RadioButton - This widget represents a mutually-exclusive selection radio button widget.
- Button -This widget represents a standard push button.
- PushButton - This widget represents a normal push button with custom styling.
- ToggleButton - This widget represents a stylish stateful button which allows the user to toggle between up and down states.
- CheckBox - This widget represents a standard check box widget. This class also serves as a base class for RadioButton.
- RadioButton - This widget represents a mutually-exclusive selection radio button widget.
- Button -This widget represents a standard push button.
- PushButton - This widget represents a normal push button with custom styling.
- ToggleButton - This widget represents a stylish stateful button which allows the user to toggle between up and down states.
- CheckBox - This widget represents a standard check box widget. This class also serves as a base class for RadioButton.
- RadioButton - This widget represents a mutually-exclusive selection radio button widget.



**Figure 4.2 Form widgets**



**Figure 4.3 Form widgets**

## GWT Compiler

- The heart of GWT is a compiler that converts Java source into JavaScript,
- i.e transforming your working Java application into an equivalent JavaScript application.
- If your GWT application compiles and runs in hosted mode.
- GWT compiles your application into JavaScript output without complaint,
- Then your application will work the same way in a web browser as it did in hosted mode

## Debugging and Deploying GWT Applications

**GWT applications can be run in two modes:**

### **1.Development mode (formerly Hosted mode):**

- The application is run as Java bytecode within the (JVM).
- This mode is typically used for development, supporting hot swapping of code(Hot swapping and Hot plugging are terms used to describe the functions of replacing computer system components without shutting down the system.) and debugging.

### **2.Production mode (formerly Web mode):**

- The application is run as pure JavaScript and HTML, compiled from the Java source.
- This mode is typically used for deployment.



## Google Web Toolkit Architecture

### GWT has three major components:

#### 1. Java-to-Javascript compiler

- Translates the Java programming language to the JavaScript programming language.

#### 2. GWT Development Mode

- Allows the developers to run and execute GWT applications in development mode.

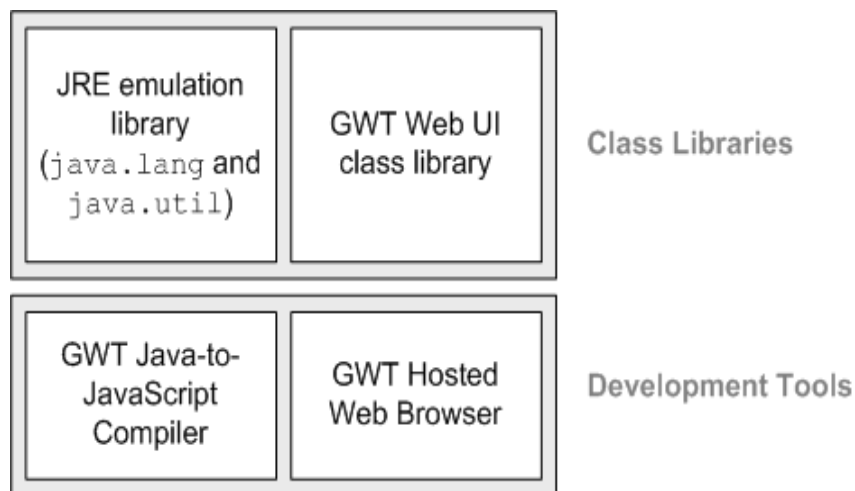
#### 3. Two Java class libraries

- (i) JRE Emulation library.
- (ii) GWT web UI class library.

**JRE Emulation library :** Google Web Toolkit includes a library that adds a subset of the Java runtime library.

The list includes *java.lang*, *java.lang.annotation*, *java.math*, *java.io*, *java.sql*, *java.util* and *java.util.logging*

**GWT UI building library :** This part of GWT consists of many subparts which includes the actual UI components, RPC support, History management, and much more shown in Figure 4.4.



**Figure 4.4 GWT UI building library**

### Google Web Toolkit Features

- Browser supports (IE, Firefox, Mozilla, Safari, and Opera)
- Browser history management
- Dynamic, reusable UI components

- Really simple RPC
- Real debugging
- Completely Open Source

## System Requirement

The GWT system requirement is shown in Figure 4.5.

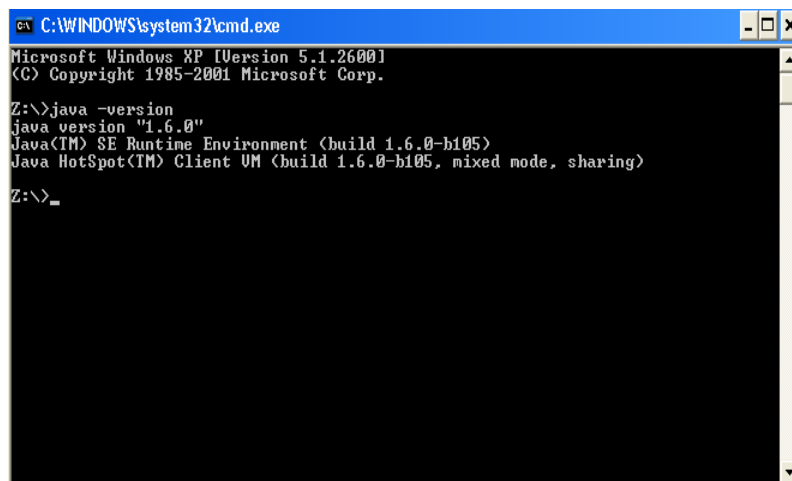
GWT requires JDK 1.6 or higher so the very first requirement is to have JDK installed in your machine.

JDK	1.6 or above.
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

**Figure 4.5 System Requirement**

Follow the given steps to setup your environment to start with GWT application development.

- Step 1 -Verify Java installation on your machine
- Now open console and execute the following java command shown in Figure 4.6.



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

Z:\>java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)

Z:\>_

```

**Figure 4.6 Command**

## Step 2 -Setup Java Development Kit (JDK):

- If you do not have Java installed then you can install the Java Software Development Kit (SDK) from Oracle's Java site: <https://www.oracle.com/java/technologies/javase-downloads.html>
- set PATH and JAVA\_HOME environment variables to refer to the directory that contains java and javac, typically java\_install\_dir/bin and java\_install\_dir respectively.
- Set the JAVA\_HOME environment variable to point to the base directory location where Java is installed on your machine. For example

## Step 3 -Setup Eclipse IDE

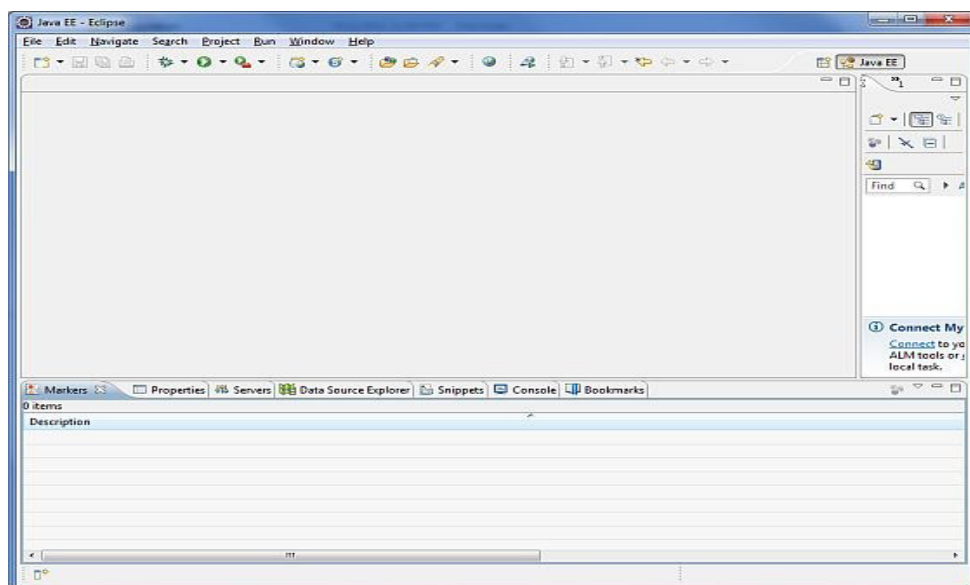
- To install Eclipse IDE, download the latest Eclipse binaries from <http://www.eclipse.org/downloads/>.
- Once you downloaded the installation, unpack the binary distribution into a convenient location.

For example in C:\eclipse on windows, or

- /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.
- Eclipse can be started by executing the following commands on windows machine, or you can simply double click on eclipse.exe
- %C:\eclipse\eclipse.exe

Eclipse can be started by executing the following commands on Unix machine:

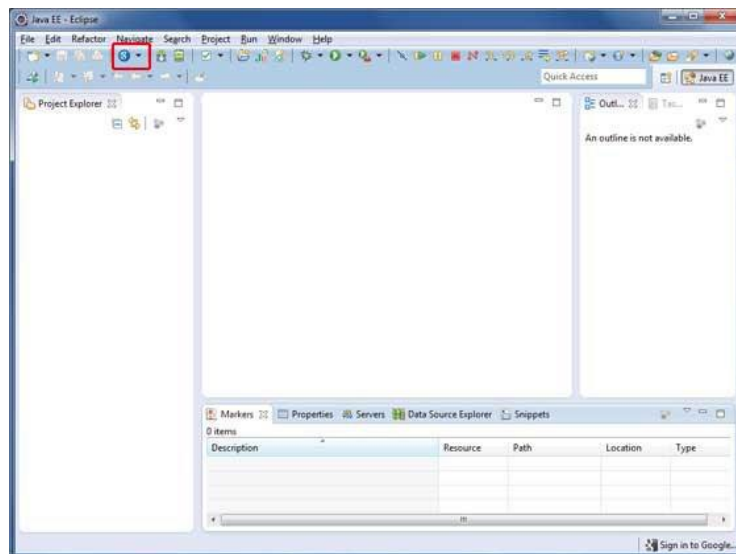
- \$/usr/local/eclipse/eclipse
- After a successful startup, if everything is fine then it should display following result in Figure : 4.7.



**Figure 4.7 Successful Startup**

**Step 4: Install GWT SDK & Plugin for Eclipse**

- **Follow the instructions given**
  - Start Eclipse, then select Help > Install New Software...
  - In the dialog that appears, enter the update site URL into the Work with text box: <http://dl.google.com/eclipse/plugin/4.2>
  - After a successful setup for the GWT plugin, if everything is fine then it should display following screen in Figure 4.8 with google icon marked with red rectangle:



**Figure 4.8 Successful Setup**

**Step 5: Setup Apache Tomcat:**

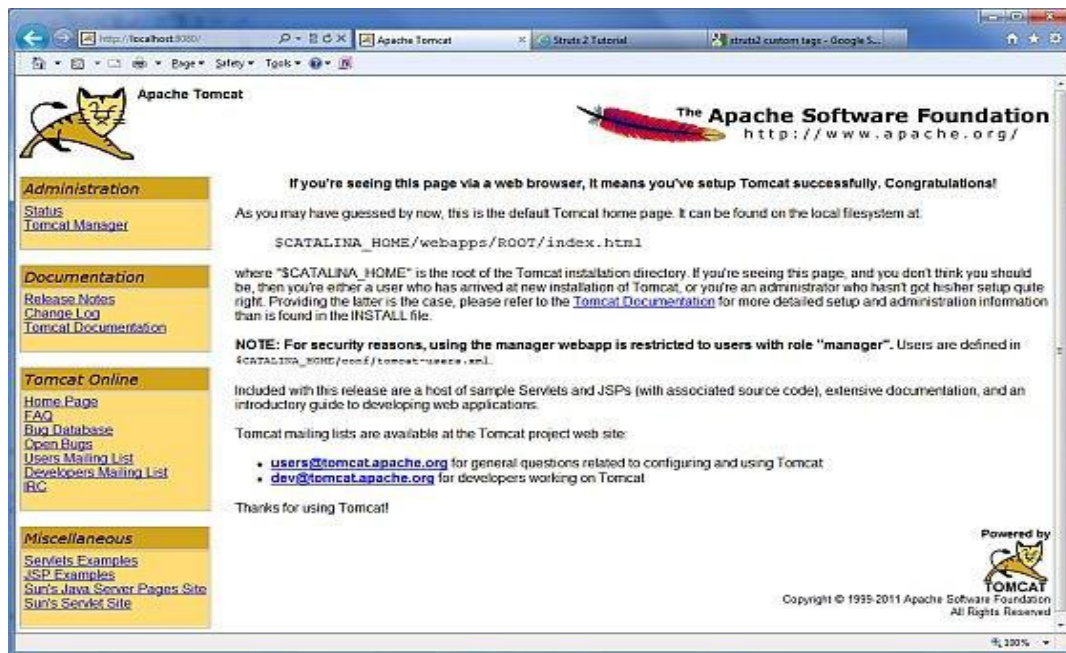
- You can download the latest version of Tomcat from <http://tomcat.apache.org/>.
- Once you downloaded the installation, unpack the binary distribution into a convenient location.

For example in C:\apache-tomcat-6.0.33 on windows,

- /usr/local/apache-tomcat-6.0.33 on Linux/Unix
- and set CATALINA\_HOME environment variable pointing to the installation locations.
- Tomcat can be started by executing the following commands on windows machine, or you can simply double click on **startup.bat**
  - %CATALINA\_HOME%\bin\startup.bat
- or
- C:\apache-tomcat-6.0.33\bin\startup.bat
  - Tomcat can be started by executing the following commands on Unix machine:

- \$CATALINA\_HOME/bin/startup.sh
- or /usr/local/apache-tomcat-6.0.33/bin/startup.sh
- **The .sh file extension is a shell script.**

After a successful startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine then it should display following result in Figure 4.9



**Figure 4.9 Successful Startup**

- Tomcat can be stopped by executing the following commands on windows machine:
  - %CATALINA\_HOME%\bin\shutdown
  - or
  - C:\apache-tomcat-5.5.29\bin\shutdown
- Tomcat can be stopped by executing the following commands on Unix machine:
  - \$CATALINA\_HOME/bin/shutdown.sh
  - or
  - /usr/local/apache-tomcat-5.5.29/bin/shutdown.sh

### Applications

A GWT application consists of following four important parts out of which last part is optional but first three parts are mandatory.

1. Module descriptors

2. Public resources
3. Client-side code
4. Server-side code

## 1. Module Descriptors

- A module descriptor is the configuration file in the form of XML which is used to configure a GWT application.
- A module descriptor file extension is \*.gwt.xml,
- where \* is the name of the application and this file should reside in the project's root.
- Following will be a default module descriptor HelloWorld.gwt.xml for a HelloWorld application is shown in Figure 4.10.

```
<?xml version="1.0" encoding="utf-8"?>
<modulerefname-to='helloworld'>
<!-- inherit the core web toolkit stuff. -->
<inheritsname='com.google.gwt.user.user' />

<!-- inherit the default gwt style sheet. -->
<inheritsname='com.google.gwt.user.theme.clean.Clean' />

<!-- specify the app entry point class. -->
<entry-pointclass='com.tutorialspoint.client.HelloWorld' />

<!-- specify the paths for translatable code -->
<sourcepath='...' />
<sourcepath='...' />

<!-- specify the paths for static files like html, css etc. -->
<publicpath='...' />
<publicpath='...' />

<!-- specify the paths for external javascript files -->
<scriptsrc='js-url' />
<scriptsrc='js-url' />

<!-- specify the paths for external style sheet files -->
<stylesheetsrc='css-url' />
<stylesheetsrc='css-url' />
</module>
```

**Figure 4.10 Default Module Descriptor**

1. <module refname-to="helloworld">
1. This provides name of the application.
- 2.<inherits name="logical-module-name" />
- This adds other gwt module in java applications.
- 3.<entry-point class="classname" />
- This specifies the name of class which will start loading the GWT Application .
4. <source path="path" />
- This specifies the names of source folders which GWT compiler will search for source compilation.

5. <public path="path" />

- The public path is the place to store resources such as CSS or images .

6. <script src="js-url" />

- Automatically inserts the external JavaScript file located at the location specified by src.

7. <stylesheet src="css-url" />

Automatically inserts the external CSS file located at the location specified by src.

## 2. Public resources

- These all files referenced by your GWT module ,such as HTML, CSS or Images.
- The location of these resources can be configured using <public path="path" /> .
- When you compile your application into JavaScript, all the files that can be found on your public path are copied to the module's output directory .

```
<html>
<head>
<title>Hello World</title>
<link rel="stylesheet"href="HelloWorld.css"/>
<script language="javascript" src="helloworld/helloworld.nocache.js">
</script> </head>
<body>
<h1>Hello World</h1>
<p>Welcome to first GWT application</p>
</body> </html>
```

Following is the sample style sheet which we have included in our host page

```
body {
text-align: center;
font-family: verdana, sans-serif; }
h1 {
font-size:2em;
font-weight: bold;
color:red;
```

```
margin:40px0px70px;
```

```
text-align: center; }
```

### 3. Client-side code

- The location of these resources can be configured using `<source path="path" />`.
- This is the actual Java code implementing the business logic of the application and that the GWT compiler translates into JavaScript .
- When a module is loaded, every entry point class is detected and its
- `EntryPoint.onModuleLoad()` method gets called.
- A sample HelloWorld Entry Point class will be as follows

```
Public class HelloWorld implements EntryPoint
{
    public void onModuleLoad()
    {
        Window.alert("Hello, World!");
    }
}
```

### 4. Server-side code

- This is the server side part of your application and its very much optional.
- If you are not doing any backend processing with-in your application then you do not need this part

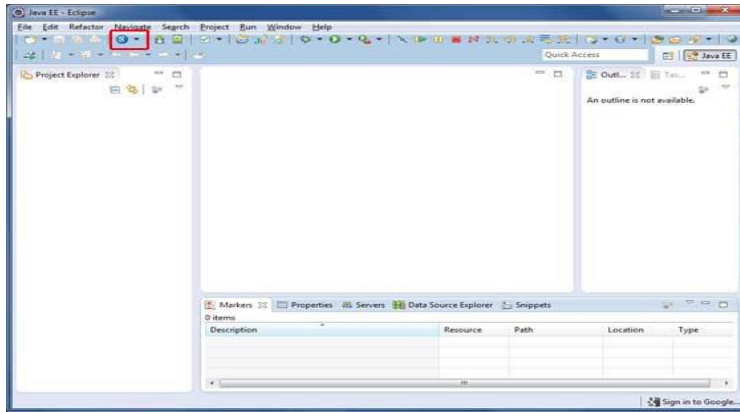
### Create Application

Let's start with a simple HelloWorld application is shown in Figure 4.11(a),Figure 4.11(b), Figure 4.11(c).

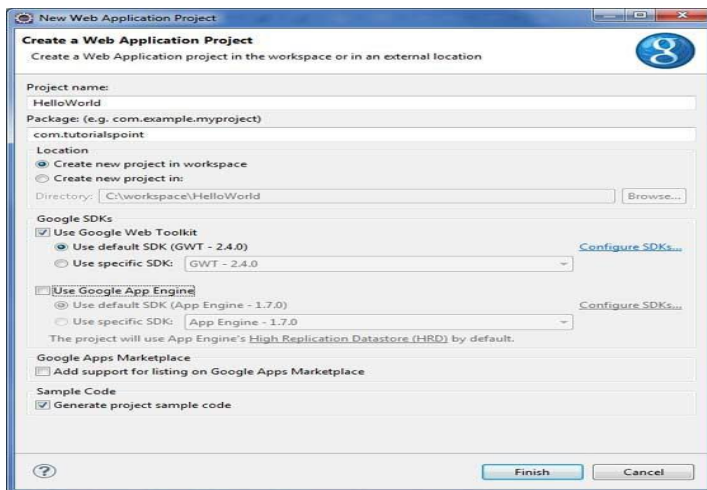
#### Step 1 -Create Project

- The first step is to create a simple Web Application Project using Eclipse IDE.
- Launch project wizard using the option
- **Google Icon > New Web Application Project....**

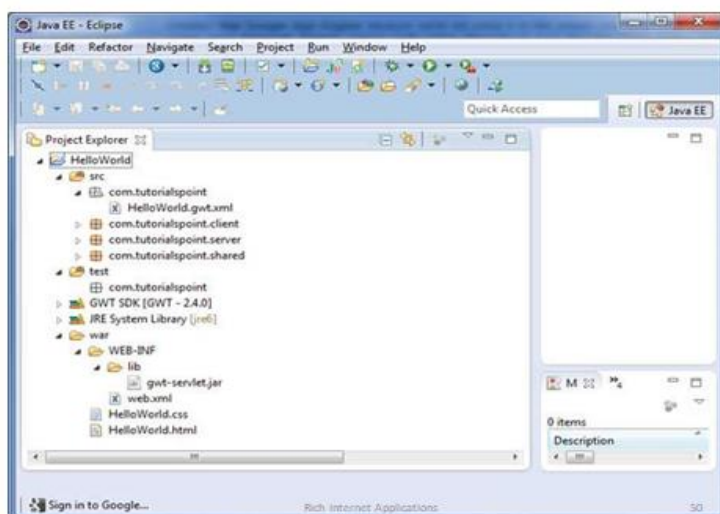




**Figure 4.11(a) Create Application-step1**



**Figure 4.11(b) Create Application-step2**



**Figure 4.11(c) Create Application-Step 3**

- **Description of all important folders**

**SRC:**

- ✓ Source code (java classes) files.
- ✓ Client folder containing the client-side specific java classes responsible for client UI display.
- ✓ Server folder containing the server-side java classes responsible for server side processing.
- ✓ Shared folder containing the java model class to transfer data from server to client and vice versa.
- ✓ HelloWorld.gwt.xml, a module descriptor file required for GWT compiler to compile the HelloWorld project.

**TEST:**

- Test code (java classes) source files.
- Client folder containing the java classes responsible to test gwt client side code.
- This is the most important part, it represents the actual deployable web application.
- WEB-INF containing compiled classes, gwt libraries, servlet libraries, HelloWorld.css, project style sheet.
- HelloWorld.html, hosts HTML which will invoke GWT UI Application.

**Step 2 -Modify Module Descriptor: HelloWorld.gwt.xml**

GWT plugin will create a default module descriptor file - *src/HelloWorld.gwt.xml* which is given below :

```
<?xml version="1.0" ?>

<modulerefname-to='helloworld'>

<!-- Inherit the core Web Toolkit stuff. -->

<inheritsname='com.google.gwt.user.User'/>

<!-- Inherit the default GWT style sheet. You can change -->

<!-- the theme of your GWT application by uncommenting -->

<!-- any one of the following lines. -->

<inheritsname='com.google.gwt.user.theme.clean.Clean'/>

<!--<inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->

<!--<inherits name='com.google.gwt.user.theme.dark.Dark'/> -->

<!-- Other module inherits -->
```

```

<!-- Specify the app entry point class. -->
<entry-pointclass='com.client.HelloWorld'/>
<!-- Specify the paths for translatable code -->
<sourcepath='client'/>
<sourcepath='shared'/>
</module>

```

### Step 3 -Modify Style Sheet: HelloWorld.css

- GWT plugin will create a default Style Sheet file *war/HelloWorld.css*.
- *Let us modify this file to keep our example at simplest level of understanding:*

```

body { text-align: center;
font-family: verdana, sans-serif; }

h1 { font-size:2em;
font-weight: bold;
color:red;
margin:40px0px70px;
text-align: center; }

```

### Step 4 -Modify Host File: HelloWorld.html

**GWT plugin will create a default HTML host file *war/HelloWorld.html*.**

*Let us modify this file to keep our example at simplest level of understanding:*

```

<html> <head> <title>Hello World</title>
<linkrel="stylesheet"href="HelloWorld.css"/>
<scriptlanguage="javascript"src="helloworld/helloworld.nocache.js">
</script></head> <body>
<h1>Hello World</h1>
<p>Welcome to first GWT application</p>
</body> </html>

```

### Step 5 -Modify Entry Point: HelloWorld.java

- GWT plugin will create a default Java file *src/com.HelloWorld.java*, which keeps an entry point for the application.
- Let us modify this file to display "Hello, World!":

```
package com.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Window;

Public class HelloWorld implements EntryPoint{

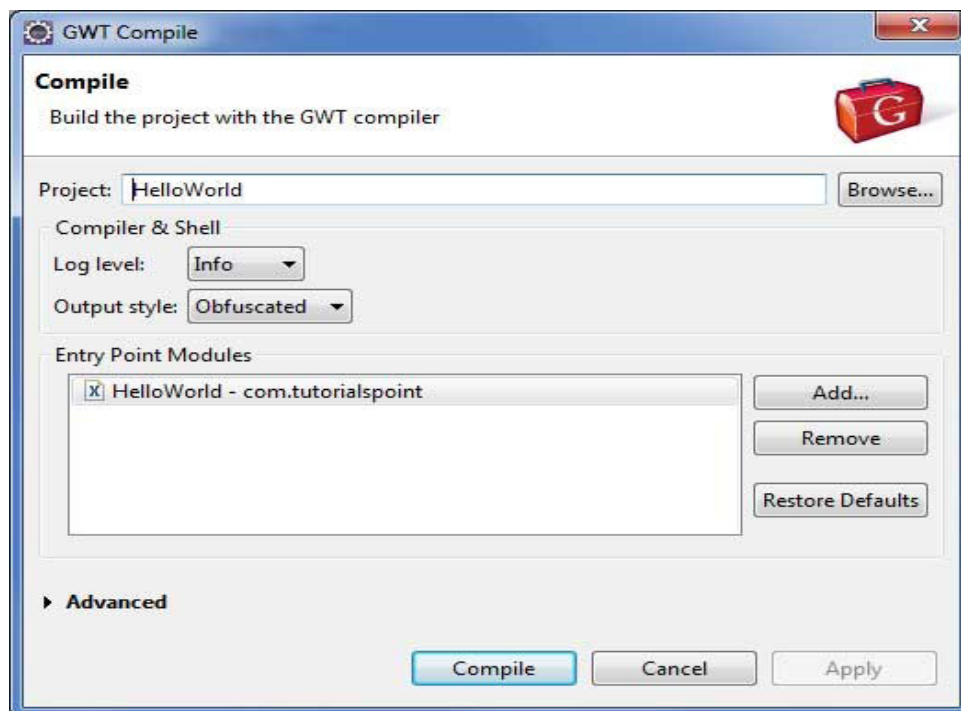
Public void onModuleLoad(){

Window.alert("Hello, World!");

} }
```

### Step 6 -Compile Application

- Once you are ready with all the changes done, its time to compile the project.
- Use the option Google Icon > GWT Compile Project...
- to launch GWT Compile dialogue box as shown below in Figure 4.12:



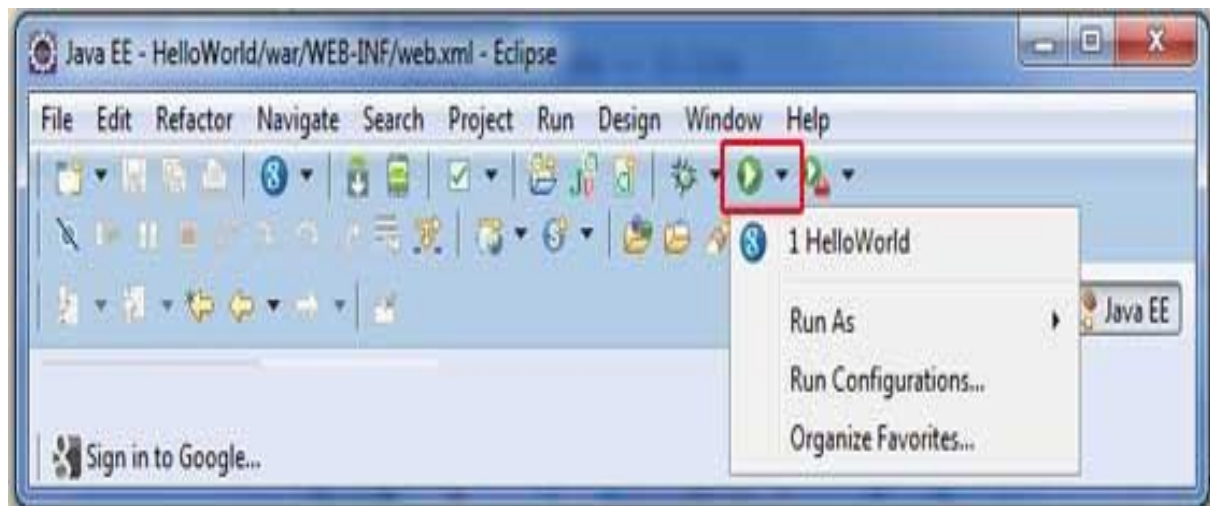
**Figure 4.12 GWT Compile Dialogue Box**

- Click Compile button. If everything goes fine, you will see following output in Eclipse console
- Compiling module com.HelloWorld

- Compiling 6 permutations
- Compiling permutation 0...
- Compiling permutation 1...
- Compiling permutation 2...
- Compiling permutation 3...
- Compiling permutation 4...
- Compiling permutation 5...
- Compile of permutations succeeded
- Linking into C:\workspace\HelloWorld\war\helloworld
- Link succeeded

### Step 7 -Run Application

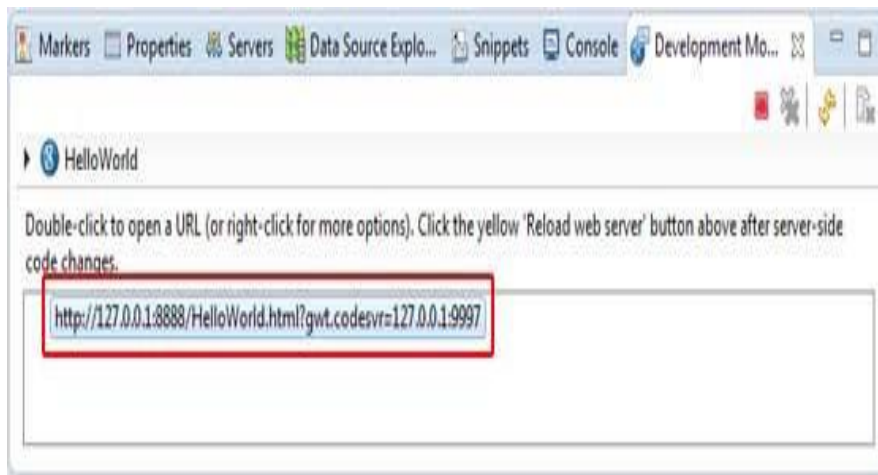
- Now click on Run application menu and select Hello World application to run the application in Figure 4.13.



**Figure 4.13 Run Application**

If everything is fine, you must see GWT Development Mode active in Eclipse containing a URL as shown below in Figure 4.14.

Double click the URL to open the GWT application.



**Figure 4.14 GWT Development Mode**

### **Deploy Application**

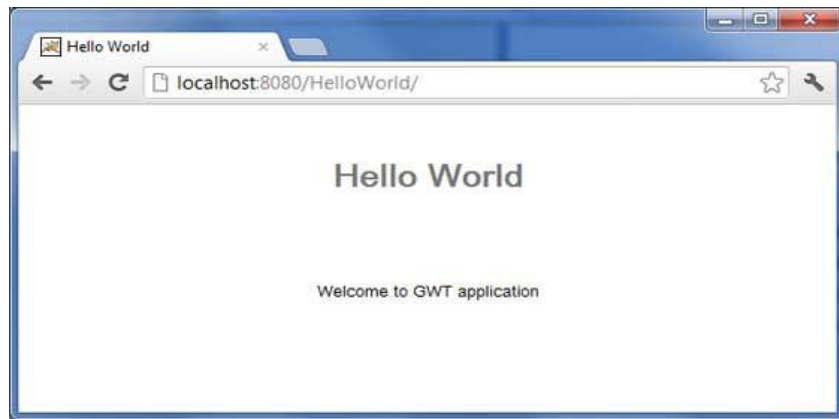
- Create WAR(Web application Archive) File
- Now our application is working fine and we are ready to export it as a war file. Follow the following steps:
- Go into your project's war directory  
C:\workspace\HelloWorld\war
- Select all the files & folders available inside war directory.
- Zip all the selected files & folders in a file called HelloWorld.zip.
- Rename HelloWorld.zip to HelloWorld.war.

### **Deploy WAR file**

- Stop the tomcat server.
- Copy the HelloWorld.war file to tomcat installation directory > webapps folder.
- Start the tomcat server.
- Look inside webapps directory, there should be a folder helloworld got created.
- Now HelloWorld.war is successfully deployed in Tomcat Webserver root.

### **Run Application –**

- Enter a url in web browser: <http://localhost:8080/HelloWorld> to launch the application in figure 4.15.
- Server name (localhost) and port (8080) may vary as per your tomcat configuration.



**Figure 4.15 Execution of Hello Application**

### **GWT Widgets**

- Button
- PushButton
- RadioButton
- CheckBox
- DatePicker
- ToggleButton
- TextBox
- PasswordTextBox
- TextArea
- Hyperlink
- ListBox
- CellList
- MenuBar
- Tree
- CellTree
- SuggestBox
- RichTextArea
- FlexTable
- Grid
- CellTable
- CellBrowser

- TabBar D
- DialogBox

### **TextBox Widget Example**

src/com.text/HelloWorld.gwt.xml.

```
<?xml version="1.0" >

<module rename-to='helloworld'>

<inherits name='com.google.gwt.user.User'/>

<inherits name='com.google.gwt.user.theme.clean.Clean'/>

<entry-point class='com.tutorialspoint.client.HelloWorld'/>

<source path='client'/>

<source path='shared'/>

</module>
```

### **war/HelloWorld.css**

```
body{

text-align: center;

font-family: verdana, sans-serif; }

h1{

font-size:2em;

font-weight: bold;

color:red;

margin:40px0px70px;

text-align: center; }

.gwt-TextArea{ color: green; }

.gwt-TextArea-readonly{ background-color: yellow; }

war/HelloWorld.html.

<html> <head>

<title>Hello World</title>

<link rel="stylesheet" href="HelloWorld.css"/>

<script language="javascript" src="helloworld/helloworld.js">

</script>
```



```
</head><body>

<h1>TextArea Widget Demonstration</h1>

<div id="gwtContainer"></div>

</body></html >
```

#### **src/com.text/HelloWorld.java**

```
package com.tutorialspoint.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextArea;
import com.google.gwt.user.client.ui.VerticalPanel;

public class HelloWorld implements EntryPoint{

    public void onModuleLoad(){

        //create textarea elements

        TextArea textArea1 =new TextArea();
        TextArea textArea2 =new TextArea();

        //set width as 10 characters

        textArea1.setCharacterWidth(20);
        textArea2.setCharacterWidth(20);

        //set height as 5 lines

        textArea1.setVisibleLines(5);
        textArea2.setVisibleLines(5);

        //add text to text area

        textArea2.setText(" Hello World! \n Be Happy! \n Stay Cool!");

        //set textbox as readonly

        textArea2.setReadOnly(true);

        // Add text boxes to the root panel.

        VerticalPanel panel =new VerticalPanel();

        panel.setSpacing(10);

        panel.add(textArea1);

        panel.add(textArea2);
```

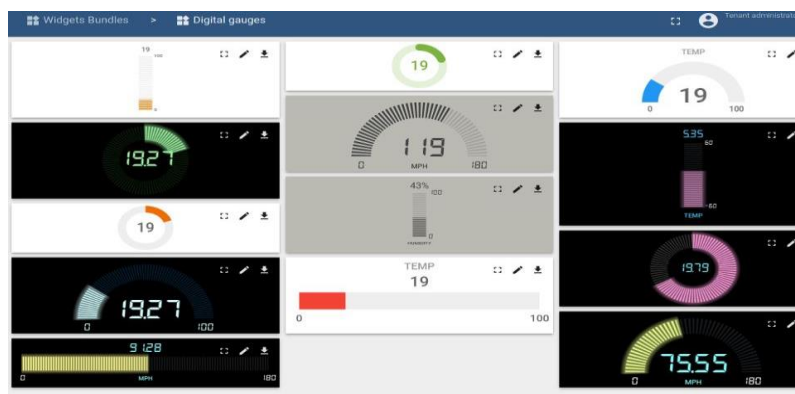
```
RootPanel.get("gwtContainer").add(panel);
}
}
```

## GWT Widget Library

- Widgets included with Service Portal can be customized to suit your own needs or as a basic code sample for you to refer to as you are building your own widgets.
- Widgets included with Service Portal can be customized to suit your own needs or as a basic code sample for you to refer to as you are building your own widgets.
- Each widget provides end-user functions such as data visualization, remote device control, alarms management and displaying static custom html content.

## Digital Gauges

Useful for visualization of temperature, humidity, speed and other integer or float values shown in Figure 4.16.



### Figure 4.16 Digital Gauges

## Analog Gauges

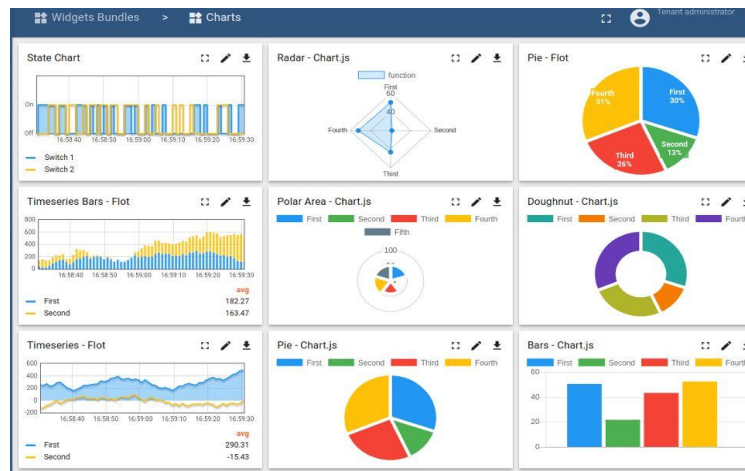
Similar to digital gauges, but have a different style in Figure 4.17.



### Figure 4.17 Analog Gauges

## Charts

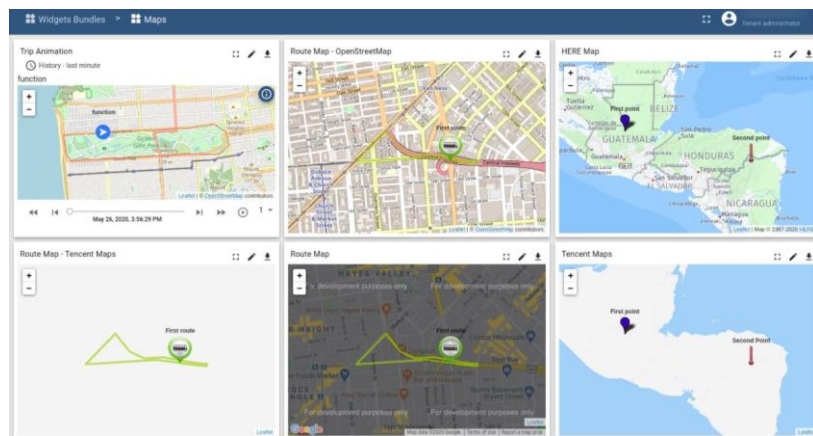
Useful for visualization of historical or real-time data with a time window in Figure 4.18.



**Figure 4.18 Charts**

## Maps widgets

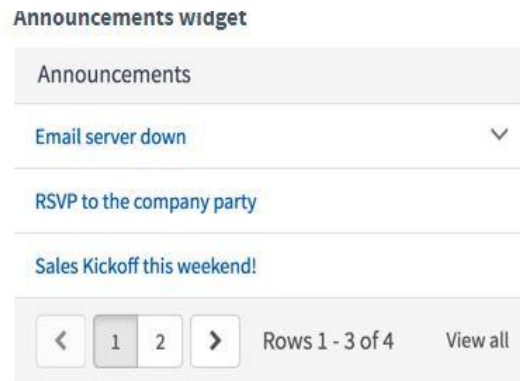
The maps widgets are shown in Figure 4.19.



**Figure 4.19 Maps Widgets**

## Announcements widget

Users can view all active announcements. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs as shown in Figure 4.20.



**Figure 4.20 Announcement Widgets**

- **Announcements widget**-Users can view all active announcements. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Approvals widget**-Users can approve or reject items directly within Service Portal. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Approval Info widget**-The Approval Info widget works in tandem with the Approval widget to display details about the approval request. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Approval Record widget**-The Approval Record widget shows the full record for an approval including the activity stream. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Breadcrumbs widget**-The breadcrumbs widget allows users to easily navigate around a portal. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Breakout Game widget**-Add a fun, interactive 404 page to pages that do not exist using the Breakout Game widget. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Calculator widget**-The calculator widget does simple calculations. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Carousel widget**-Showcase specific items in your catalog using a scrolling list of images in the carousel widget. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Change Password widget**-Users can change their passwords using the Change Password widget.
- **Cool Clock widget**-Show different times around the world using the Cool Clock widget. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.

- **Data table from instance definition widget**-Display a filtered list on your portal using the data table from instance definition widget. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Data Table from URL definition widget**-The Data Table from URL definition widget displays the table you select from the list. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Form widget**-The form widget is a platform form within the Service Portal UI with a few differences. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Header menu widget**-The Header Menu widget controls which options appear in the page header. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Hello World widgets**-The Hello World widgets are included with Service Portal as examples of how to use and create widgets. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **HTML widget**-Use the HTML widget to directly inject HTML, text, lists, or content in general into a page. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Icon Link widget**-Link to any other item. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Icon menu list widget**-A simple list with a glyph icon next to each link. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Language Switch widget**-Add the Language Switch widget to a landing or homepage to allow your users to change the language of the page. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Link button widget**-The Link Button widget is a button you can nest in any other widget that links to another destination. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Login widget**-The login widget controls user access to your site. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **My Requests widget**-The My Requests widget stores all of your open requests in one place. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **My Requests widget**-Use the My Requests widget (my-requests-v2) to enable requesters to view open or closed requests in Service Portal. Requests, incidents, and tasks are displayed in a single view that is based on the filter conditions and display settings in the My Request Filter module. For information on defining filters for this module, refer to the Related information section.

- **Organization Chart widget**-The Organization Chart widget shows employees in a tree structure relative to their manager. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Simple List widget**-The Simple List widget can be used to display any list in the system within Service Portal. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Sample Footer widget**-The Sample Footer widget is an example of a footer you can use in your portal. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Ticket Attachments widget**-Use the attachment widget to attach items to tickets. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Ticket Conversations widget**-Record of ticket items. Users can use this widget to communicate back and forth with the fulfiller and the receiver. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Ticket Fields widget**-The Ticket Fields widget displays information about a request that a user has made. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **Ticket Location widget**-Share your location in a ticket. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.
- **User Profile widget**-Display user profile information. You can use this base system widget as-is in your Service Portal or clone it to suit your own business needs.

## References

1. Federico Kerek , “ Essential GWT: Building for the Web with Google Web Toolkit 2 ”, Addison-Wesley Professional,2010.



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT – V-Rich Internet Applications – SCS1401**

## **V. Building Ria With GWT**

**GWT Panels and Layouts - Event Handling - Internationalisation - Advanced GWT - RPC and AJAX – Writing a service implementation - Handling browser back button functionality - MVP Design pattern.**

### **GWT Layout Panels**

- Layout Panel helps us to design the user interface of the panels.
- This panel helps to fit all the content inside the windows.
- Every Panel widget inherits properties from Panel class which in turn inherits properties from Widget class and which in turn inherits properties from UIObject class.

### **Types of Layout Panels**

1. Flow Panel
2. Horizontal Panel
3. Vertical Panel
4. Horizontal Split Panel
5. Vertical Split Panel
6. Flex Table
7. Grid
8. Deck Panel
9. Dock Panel
10. HTML Panel
11. Tab Panel
12. Composite
13. Simple Panel
14. Scroll Panel
15. Focus Panel
16. Form Panel
17. Popup Panel
18. Dialog Box



## Flow Panel

- This widget represents a panel that formats its child widgets using the default HTML layout behavior shown in Figure 5.1.

### Class Declaration

```
public class FlowPanel extends ComplexPanel
    implements InsertPanel.ForIsWidget
```

### Constructor

```
FlowPanel()
```

### Class Methods

- **void add(Widget w)**-Adds a new child widget to the panel.
- **void clear()**-Removes all child widgets.
- **void insert(Widget w, int beforeIndex)**

### EXAMPLE

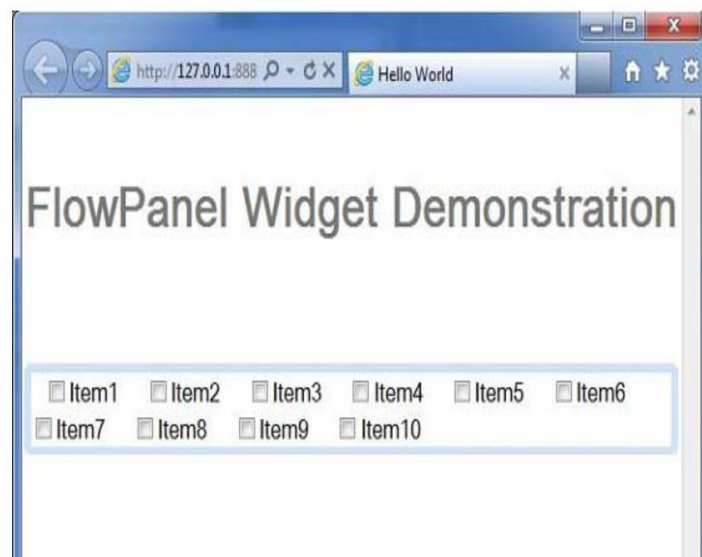


Figure 5.1 Flow Layout

## HorizontalPanel

- The **HorizontalPanel** widget represents a panel that lays all of its widgets out in a single horizontal column in figure 5.2.

### Class Declaration

```
public class HorizontalPanel extends CellPanel
    implements HasAlignment, InsertPanel.ForIsWidget
```

## Class Constructors

### HorizontalPanel()

## Class Methods

- **void add(Widget w)**- Adds a child widget.
- **HasHorizontalAlignment.HorizontalAlignmentConstant**  
**getHorizontalAlignment()**- Gets the horizontal alignment.
- **HasVerticalAlignment.VerticalAlignmentConstant**  
**getVerticalAlignment()**-Gets the vertical alignment.
- **void insert(Widget w, int beforeIndex)**- Inserts a child widget before the specified index.
- **boolean remove(Widget w)**- Removes a child widget.



**Figure 5.2 Horizontal Layout**

## Vertical Panel

- The **VerticalPanel** widget represents a panel that lays all of its widgets out in a single vertical in Figure 5.3.
- row.

## Class Declaration

```
public class VerticalPanel extends CellPanel
    implements HasAlignment, InsertPanel.ForIsWidget
```

## Class Constructors

### VerticalPanel()

## Class Methods

- **void add(Widget w)**-Adds a child widget.
- **boolean remove(Widget w)**-Removes a child widget.



Figure 5.3 Vertical Panel

## Horizontal Split Panel

- The **HorizontalSplitPanel** widget represents a panel that arranges two widgets in a single horizontal row and allows the user to interactively change the proportion of the width dedicated to each of the two widgets.

### Class Declaration

```
public final class HorizontalSplitPanel extends Panel
```

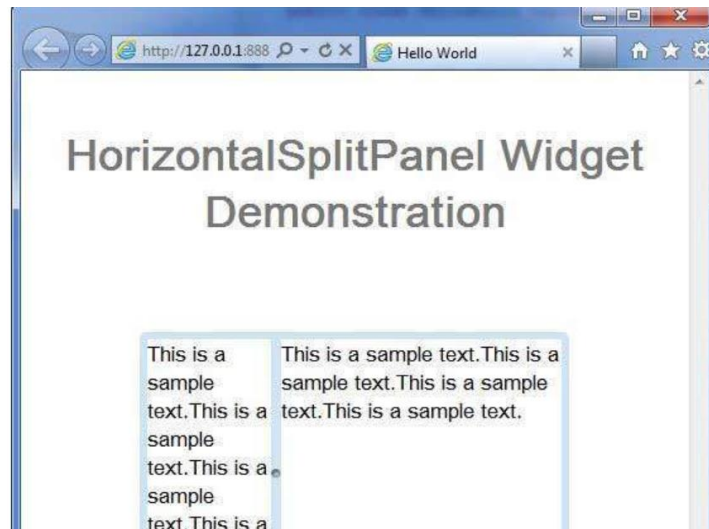
### Constructors

```
HorizontalSplitPanel()
```

```
HorizontalSplitPanel(HorizontalSplitPanel.Resources resources)
```

### Class Methods

- **void add(Widget w)**
- **Widget getLeftWidget()**-Gets the widget in the left side of the panel.
- **Widget getRightWidget()**-Gets the widget in the right side of the panel.
- **boolean isResizing()**-Indicates whether the split panel is being resized.
- **boolean remove(Widget widget)**-Removes a child widget.
- **void setLeftWidget(Widget w)**-Sets the widget in the left side of the panel.
- **void setRightWidget(Widget w)**-Sets the widget in the right side of the panel.



**Figure 5.4 Horizontal Split Panel**

### VerticalSplit Panel

- The **VerticalSplitPanel** widget represents a panel that arranges two widgets in a single vertical column and allows the user to interactively change the proportion of the height dedicated to each of the two widgets.
- Widgets contained within a VerticalSplitterPanel will be automatically decorated with scrollbars when necessary.

#### Class Declaration

```
public final class VerticalSplitPanel extends Panel
```

### FlexTable

- The **FlexTable** widget represents a flexible table that creates cells on demand.
- It can be jagged (that is, each row can contain a different number of cells) and individual cells can be set to span multiple rows or columns.

#### Class Declaration

```
public class FlexTable extends HTMLTable
```

#### Class Methods

**void addCell(int row)**-Appends a cell to the specified row.

**int getCellCount(int row)**-Gets the number of cells on a given row.

**int getRowCount()**-Gets the number of rows.

**void insertCell(int beforeRow, int beforeColumn)**-Inserts a cell into the FlexTable.

**int insertRow(int beforeRow)**-Inserts a row into the FlexTable.

## Grid

- This widget represents a rectangular grid that can contain text, html, or a child Widget within its cells.
- It must be resized explicitly to the desired number of rows and columns.

### Class Declaration

```
public class Grid extends HTMLTable
```

### Constructors

```
Grid()
```

```
Grid(int rows, int columns)
```

### Class Methods

- **boolean clearCell(int row, int column)**-Replaces the contents of the specified cell with a single space.
- **protected Element createCell()**-Creates a new, empty cell.
- **int getCellCount(int row)**-Return number of columns.
- **int getColumnCount()**-Gets the number of columns in this grid.
- **int getRowCount()**-Return number of rows.
- **void removeRow(int row)**-Removes the specified row from the table.
- **void resize(int rows, int columns)**-Resizes the grid.

## Deck Panel

- Panel that displays all of its child widgets in a 'deck', where only one can be visible at a time. It is used by TabPanel.

### Class Declaration

```
public class DeckPanel extends ComplexPanel  
    implements HasAnimation, InsertPanel.ForIsWidget
```

### Class Methods

```
void add(Widget w)-Adds a child widget.
```

```
int getVisibleWidget()-Gets the index of the currently-visible widget.
```

## DockPanel

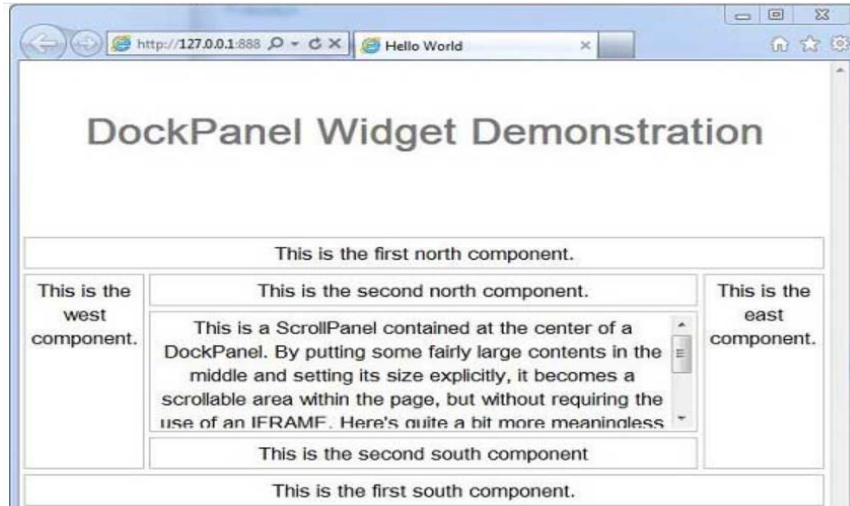
- This widget represents a panel that lays its child widgets out "docked" at its outer edges, and allows its last widget to take up the remaining space in its center is shown in Figure 5.5.

### Class Declaration

- `public class DockPanel extends CellPanel implements HasAlignment`

### Class Methods

**`void add(Widget widget, DockPanel.DockLayoutConstant direction)`**- Adds a widget to the specified edge of the dock.



**Figure 5.5 Dock Panel**

### HTMLPanel

- This widget represents a panel that contains HTML, and which can attach child widgets to identified elements within that HTML is shown in Figure 5.6.

#### Class

`public class HTMLPanel extends ComplexPanel`

#### Class Methods

- **`void add(Widget widget, Element elem)`**
- **`void addAndReplaceElement(Widget widget, Element toReplace)`**-Adds a child widget to the panel, replacing the HTML element.
- **`void addAndReplaceElement(Widget widget, java.lang.String id)`**-Adds a child widget to the panel, replacing the HTML element specified by a given id.



**Figure 5.6 HTML Panel**

## **TabPanel**

- The **TabPanel** widget represents panel that represents a tabbed set of pages, each of which contains another widget.
- Its child widgets are shown as the user selects the various tabs associated with them.

### **Class Declaration**

```
public class TabPanel extends Composite
```

### **Methods**

- **void add(Widget w)**
- **void add(Widget w, Widget tabWidget)**
- **TabBar getTabBar()**

## **Composite Widget**

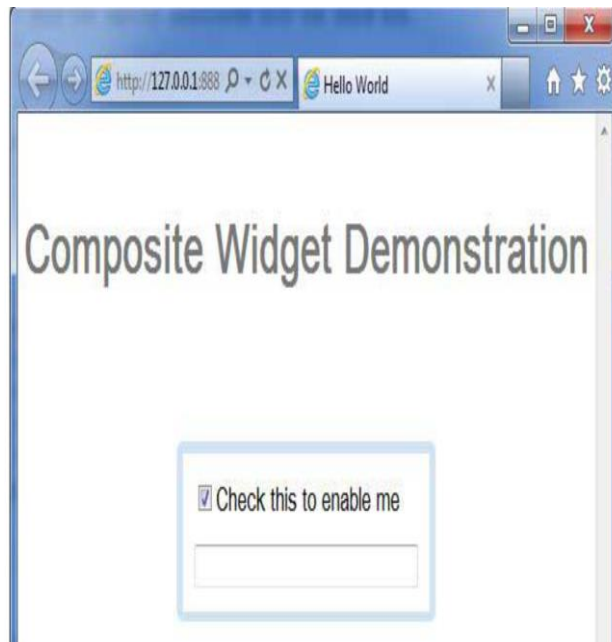
- The **Composite** widget is a type of widget that can wrap another widget, hiding the wrapped widget's methods is shown in Figure 5.7.

### **Class**

```
public abstract class Composite extends Widget
```

### **Class Methods**

- **protected Widget getWidget()**-Provides subclasses access to the topmost widget that defines this composite.
- **protected void initWidget(Widget widget)**-Sets the widget to be wrapped by the composite.
- **boolean isAttached()**-Determines whether this widget is currently attached to the browser's document.



**Figure 5.7 Composite Widget**

### **SimplePanel Widget**

- The **SimplePanel** widget represents a base class for panels that contain only one widget in Figure 5.8.

#### **Class Declaration**

```
public class SimplePanel extends Panel implements HasOneWidget
```

#### **Class Methods**

- **void add(Widget w)**-Adds a widget to this panel.
- **Widget getWidget()**-Gets the panel's child widget.
- **boolean remove(Widget w)**-Removes a child widget.
- **void setWidget(IsWidget w)**-Set the only widget of the receiver, replacing the previous widget if there was one.



**Figure 5.8 Simple Panel Widget**



## ScrollPane Widget

- The **ScrollPane** widget represents a simple panel that wraps its contents in a scrollable area in figure 5.9.

### Class Declaration

- `public class ScrollPanel extends SimplePanel` implements `SourcesScrollEvents`, `HasScrollHandlers`, `RequiresResize`, `ProvidesResize`

### Class Methods

- **`void ensureVisible(UIObject item)`**-Ensures that the specified item is visible, by adjusting the panel's scroll position.
- **`int getHorizontalScrollPosition()`**-Gets the horizontal scroll position.
- **`int getScrollPosition()`**-Gets the vertical scroll position.
- **`void scrollToBottom()`**-Scroll to the bottom of this panel.
- **`void scrollToLeft()`**-Scroll to the far left of this panel.
- **`void scrollToRight()`**-Scroll to the far right of this panel.



**Figure 5.9 Scroll Panel Widget**

## FocusPanel Widget

- The **FocusPanel** widget represents a simple panel that makes its contents focusable, and adds the ability to catch mouse and keyboard events in Figure 5.10.

### Class Declaration

```
public class FocusPanel extends SimplePanel
```

### Class Methods

- **`void addClickListener(ClickListener listener)`**

**void addMouseListener(MouseListener listener)**



**Figure 5.10 FocusPanel Widget**

### **FormPanel Widget**

- The **FormPanel** widget represents a panel that wraps its contents in an HTML `<FORM>` element in Figure 5.11.

#### **Class**

```
public class FormPanel extends SimplePanel
```

#### **Class Methods**

- **void add Form Handler (FormHandler handler)**
- **boolean onFormSubmit()**



**Figure 5.11 FormPanel Widget**

## PopupPanel

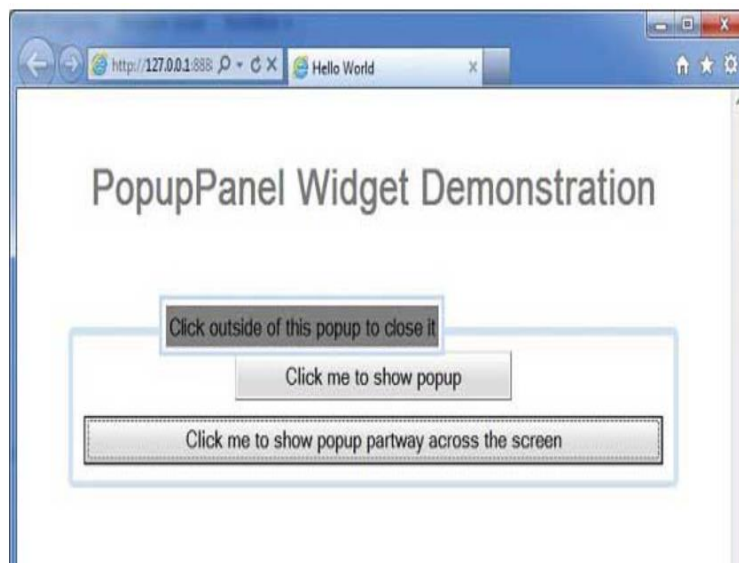
- The **PopupPanel** widget represents a panel that can **pop up** over other widgets Figure 5.12.

### Class Declaration

- `public class PopupPanel extends SimplePanel`

### Class Methods

- `void addPopupListener(PopupListener listener)`
- `void center()`



**Figure 5.12 Popup Panel**

## DialogBox Widget

- The **DialogBox** widget represents a form of popup that has a caption area at the top and can be dragged by the user shown in Figure 5.13.

### Class

- `public class DialogBox extends DecoratedPopupPanel`

### Methods

- `void onMouseMove(Widget sender, int x, int y)`
- `void onMouseUp(Widget sender, int x, int y)`



**Figure 5.13 DialogBox Widget**

## Event Handling in GWT

- GWT provides a list of interfaces corresponding to various possible events.
- A listener interface defines one or more methods that the widget calls to announce an event.
- For example, the **Button** class publishes **click events** so you will have to write a class to implement *ClickHandler* to handle **click** event.

## Event Handler Interfaces

- All GWT event handlers have been extended from *EventHandler* interface and each handler has only a single method with a single argument.
- Each **event** object have a number of methods to manipulate the passed event object.
  - ```
public class MyClickHandler implements ClickHandler {    @Override
    public void onClick(ClickEvent event) {    Window.alert("Hello World!");
    } }
```
- **BlurHandler-void on Blur(Blur Event event);**
- **ChangeHandler-void on Change(ChangeEvent event);**-Called when a change event is fired.
- **ClickHandler-void on Click(ClickEvent event);**
- **CloseHandler-void on Close(CloseEvent<T> event);**-Called when CloseEvent is fired.
- **Context Menu Handler-void on Context Menu(Context Menu Event event);**
- **Double Click Handler-void on Double Click(Double Click Event event);**-Called when a Double Click Event is fired.
- **Error Handler-void on Error(Error Event event);**-Called when Error Event is fired.
- **Focus Handler-void on Focus(Focus Event event);**
- **FormPanel.SubmitHandler-void on Submit(Form Panel.Submit Event event);**- Fired when the form is submitted.

- **Key Down Handler-void on Key Down(Key Down Event event);**-Called when KeyDownEvent is fired.
- **KeyPressHandler-void on KeyPress(KeyPressEvent event);**-Called when KeyPressEvent is fired.
- **KeyUpHandler-void on KeyUp(KeyUpEvent event);**-Called when KeyUpEvent is fired.
- **LoadHandler-void on Load(LoadEvent event);**-Called when LoadEvent is fired.
- **MouseDownHandler-void on MouseDown(MouseDownEvent event);**-Called when MouseDown is fired.

### **GWT Internationalization**

- It is similar to Java programming language, where internationalization is implemented by means of Resource Bundles, Where **.properties** file is created for each locale that needs to be supported.
- Internationalization is changing the language of the text based on the locale. For example, the browser should display the website content in Hindi for a user sitting in India and French for the user accessing the website from France.
- **Types of Internationalization Techniques**
  1. Static String Internationalization
  2. Dynamic String Internationalization
  3. Localizable Interface

### **Static String Internationalization**

- It is a good technique for translating both constant and parameterized strings.
- It is the simplest technique to implement as it requires very less over head.
- It uses standard Java properties files to store translated strings and parameterized messages.

### **Dynamic String Internationalization**

- Dynamic string internationalization is slower but more flexible than static string internationalization.
- Applications using this technique look **like** localized strings in the module's home page. Due to this technique they do not need to be recompiled when you add a new locale.

### **Localizable Interface**

- It is the most powerful technique to implement the interface.
- It is an advanced internationalization technique that is **used rarely**.

- We require advance level to implement Localizable interface **for** simple string substitution. It also creates localized versions of custom types.

### **Advanced GWT**

- Advanced GWT Components is an extension of the standard Google Web Toolkit library.
- It allows making rich web interfaces extremely quickly even if you're not skilled in DHTML and JavaScript programming.
- Currently the library supports such popular browsers like Internet Explorer, Firefox, Safari, Opera and Chrome.

### **Widgets**

- Editable (updatable) Grid
- Hierarchical Grid
- Tree Grid
- Advanced FlexTable
- SimpleGrid
- AdvancedTabPanel
- SingleBorder
- RoundCornerBorder
- Pager
- Grid Toolbar
- Grid Panel
- Master-Detail Panel
- Date Picker
- Combo Box
- Suggestion Box

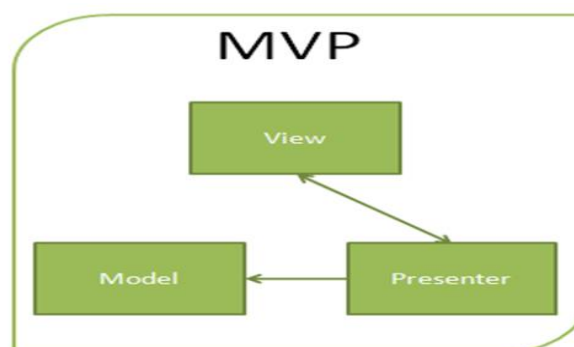
### **Advanced GWT features**

- Improved browser independent table body scrolling
- Editable grids
- Hierarchical and master-detail representations of complicated data models
- Multiple and single row selections
- Client side paging and sorting
- Server side data loading

- Client and server side content rendering
- Full keyboard control
- Flexible tab panels
- Customizable borders API
- Localization and internationalization
- <thead> HTML tag support in tables
- <tfoot> HTML tag support in tables
- Splitting to view and model (MVP design)
- Data model events
- Lazy rendering for large drop down lists
- Customizable event management and rendering
- GWT 1.6.4, 1.7.1, 2.0.x, 2.1.x, 2.2.x, 2.3.x, 2.4.x, 2.5.x or higher compatibility
- No dependencies to other third-party libraries
- Theme runtime switching for non-styled widgets
- Latest releases are available in the Maven Central Repository

### MVP Design Pattern

- MVP (Model View Presenter) is a design pattern which allows the application developing in GWT to follow MVP architecture in Figure 5.14.
- MVP provides the solution of the problem of complexity for developing application.
- Application development is complex as many developers working on same code due to which all follow same design pattern.



**Figure 5.14 MVP**

- Model: This segment model consists of data only. It holds within the business object which is to be manipulated and calculated according to application need.

- View: It only consists of view i.e. display the data which is given by presenter. It provides reusability of view code as we can swap the new view very easily. It only deals with the HTML and CSS which also helps in separate testing.
- Presenter: It contains all the logic which is to be implemented in the application development. It communicates with model as well as view. It is complete distinct in operation which provides separate JUnit testing.

## MVP Vs MVC

The difference between MVP and MVC is discussed in Table 5.1.

| MVP (Model View Presenter)                                | MVC (Model View Controller)                                          |
|-----------------------------------------------------------|----------------------------------------------------------------------|
| It is advance form of MVC                                 | It is the basic method to separate project structure.                |
| In this View handles user gesture and call presenter.     | In this controller handles user gesture and commands model.          |
| View is dumb i.e. all interaction goes through Presenter. | In this view has some intelligence. It can query the model directly. |
| It highly supports unit testing.                          | It provides limited support to unit testing.                         |
| It has high degree of loose coupling.                     | It has fairly loose coupling.                                        |
| In this presenter will update its associated view.        | It identifies which view to update.                                  |

**Table 5.1 MVP vs MVC**

## MVP Design Pattern – Benefits

- MVP is a design pattern that breaks your app up into the components Model, View and Presenter.
- The MVP pattern is extremely useful when building large, web-based applications with GWT.
- It helps to make code more readable, and more maintainable.
- It also makes it much easier to implement new features, optimizations, and automated testing.

## Model

- Houses all of the data objects that are presented and acted upon within your UI.



- The number and granularity of models is a design decision. In our PhotoApp, we will have one very simple model:
- PhotoDetails - holds data about a photo, including the thumbnail URL, original URL, title, description, tags, and so on.

### **View**

- These are the UI components that display model data and send user commands back to the presenter component.
- It is not a requirement to have a view per model. You may have a view that uses several models, or several views for one model. We will keep things simple for our Photo Application, and will have three views:
  - WelcomeView – A very simple welcome page.
  - PhotoListView - Displays a list of thumbnail photos and their title.
  - PhotoDetailsView - displays the photo together with title and other data and allows the user to change some of those details.

### **Presenter**

- The presenter will hold the complex application and business logic used to drive UIs and changes to the model. It also has the code to handle changes in the UI that the view has sent back.
- Usually for each view, there will be an associated presenter. In our photo application, this means we will have the following three presenters:
- WelcomePresenter - pushes the welcome screen in front of the user, and handles the jump to PhotoListView.
- PhotoListPresenter - drives the thumbnail view.
- PhotoDetailsPresenter - drives the view of the original photo.

### **Creating Views**

- Remember that our view should have no application logic in it, at all. It should be just UI components that the presenter can access to set or get values from.
- All of our views will be implemented as three separate items: a generic interface, a specific interface and an implementation.

```
public interface View extends IsWidget{
    void setPresenter(PhotoDetailsPresenter presenter);
}
```

### **Implementing the views**

- Take the detailed view, each implementation implements the setPresenter method

```
public void setPresenter(PhotoDetailsPresenter presenter) {
    this.presenter = presenter;
}
```

### **Presenters**

- Presenters are where all the application logic sits and will have no UI components.
- In a similar way to views, we provide a generic presenter interface, a specific one, and an implementation for each presenter.

### **Handling browser back button functionality**

- GWT History mechanism is similar to the Ajax history implementations such as RSH (Really Simple History).
- Basic idea is to track application internal state in the URL fragment identifier.

### **Main advantages of this mechanism are:**

- It provides browser history reliable.
- It provides good feedback to the user.
- It is bookmarkable i.e., the user can create a bookmark to the current state and save it or can email it etc.

### **GWT History Syntax**

```
public class History extends java.lang.Object
```

### **GWT History Tokens**

- A token is simply a string that the application can parse to return to a particular state.
- This token will be saved in browser history as a URL fragment (in the location bar, after the "#"), and this fragment is passed back to the application when the user goes back or forward in history, or follows a link.
- Example: History token name - javatpoint.
- <http://www.example.com/com.example.gwt.HistoryExample/HistoryExample.html#javatpoint>

### **GWT Hyperlink Widgets**

- Hyperlinks are convenient to use to incorporate history support into an application. Hyperlink widgets are GWT widgets that look like regular HTML anchors. You can associate a history token with the Hyperlink, and when it is clicked, the history token is automatically added to the browser history stack. The `History.newItem(token)` step is done automatically.
- Handling an `onValueChange()` callback

- The first step of handling the `onValueChanged()` callback method in a `ValueChangeHandler` is to get the new history token with `ValueChangeEvent.getValue()` then we will parse the token. Once the token is parsed, we can reset the state of the application.
- When the `onValueChanged()` method is invoked, application handles two cases:
- The application was just started and was passed a history token.

The application is already running and was passed a history token.

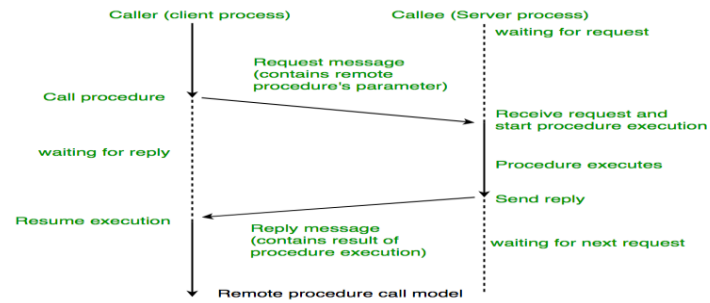
## **AJAX and RPC**

### **Server-side Code**

- Everything that happens within your web server is referred to as server-side processing.
- When your application running in the user's browser needs to interact with your server (for example, to load or save data), it makes an HTTP request across the network using a remote procedure call (RPC).
- While processing an RPC, your server is executing server-side code.
- GWT provides an RPC mechanism based on Java Servlets to provide access to server-side resources.
- This mechanism includes generation of efficient client-side and server-side code to serialize objects across the network using deferred binding.

### **Remote Procedure Calls**

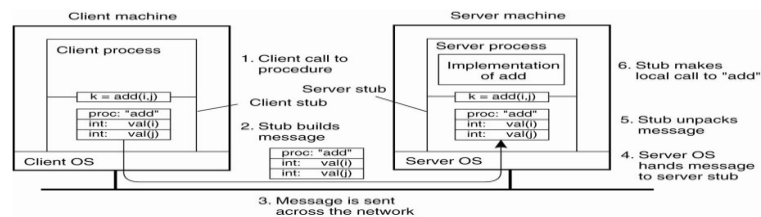
- A fundamental difference between AJAX applications and traditional HTML web applications is that AJAX applications do not need to fetch new HTML pages while they execute.
- Because AJAX pages actually run more like applications within the browser, there is no need to request new HTML from the server to make user interface updates.
- However, like all client/server applications, AJAX applications usually do need to fetch data from the server as they execute.
- The mechanism for interacting with a server across a network is called making a remote procedure call (RPC), also sometimes referred to as a server call is shown in Figure 5.15 and Figure 5.16.



**Figure 5.15 RPC**

### Example of an RPC

No message passing at all is visible to the programmer.



**Figure 5.16 Example of an RPC**

- GWT RPC makes it easy for the client and server to pass Java objects back and forth over HTTP.
- When used properly, RPCs give you the opportunity to move all of your UI logic to the client, resulting in greatly improved performance, reduced bandwidth, reduced web server load, and a pleasantly fluid user experience.
- The server-side code that gets invoked from the client is often referred to as a service, so the act of making a remote procedure call is sometimes referred to as invoking a service

### Creating Services

- In order to define your RPC interface, you need to:
- Define an interface for your service that extends `RemoteService` and lists all your RPC methods.
- Define a class to implement the server-side code that extends `RemoteServiceServlet` and implements the interface you created above.
- Define an asynchronous interface to your service to be called from the client-side code

### Synchronous Interface

- To begin developing a new service interface, create a client-side Java interface that extends the `RemoteService` tag interface.

```
package com.example.foo.client;
```

```
import com.google.gwt.user.client.rpc.RemoteService;

public interface MyService extends RemoteService {

    public String myMethod(String s);

}
```

- Any implementation of this service on the server-side must extend RemoteServiceServlet and implement this service interface.

```
package com.example.foo.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import com.example.foo.client.MyService;

public class MyServiceImpl extends RemoteServiceServlet
implements

    MyService {

    public String myMethod(String s) {

        return s;

    }
}
```

- It is not possible to call this version of the RPC directly from the client.
- You must create an asynchronous interface to all your services as shown below.

### **Asynchronous Interfaces**

- Before you can actually attempt to make a remote call from the client, you must create another client interface, an asynchronous one, based on your original service interface.
- Continuing with the example above, create a new interface in the client subpackage:

```
package com.example.foo.client;

interface MyServiceAsync {

    public void myMethod(String s, AsyncCallback<String> callback);

}
```

- An interaction is synchronous if the caller of a method must wait for the method's work to complete before the caller can continue its processing.
- An interaction is asynchronous if the called method returns immediately, allowing the caller to continue its processing without delay.

### **References**

1. Federico Kerek , “ Essential GWT: Building for the Web with Google Web Toolkit 2 ”, Addison-Wesley Professional,2010.